VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Ovládání robotů pro fotbal robotů

Robots Controling for Robot Soccer

2011

Roman Husák

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2011 .....

Touto cestou bych rád poděkoval vedoucímu diplomové práce Ing. Janu Martinovičovi, Ph.D za konzultace a rady při psaní mé diplomové práce. Dále pak bych poděkoval Ing. Václavu Svatoňovi za pomoc při řešení problémů, které nastaly v průběhu zpracování práce a Bc. Danielu Štrbovi za jeho ochotu a čas strávený při experimentech a řešení určitých algoritmů.

Abstrakt

Tato diplomová práce má za cíl ovládání pohybu naších robotů pro hru fotbal robotů. První fáze je zaměřená na počáteční inicializaci, která se především zabývá počítačem, vysílačem a robotem. Jsou zde popsány postupy nastavení jednotlivých periférií, aby nedocházelo k chybám v důsledku špatné konfigurace. Na robotech byly provedeny rozsáhle experimenty, jejichž výsledky a zhodnocení jsou součástí textu. Práce popisuje výběr taktiky pro jednotlivé roboty v závislosti na reálných datech z hřiště. Tato data zpracovávají vlastnosti, jež popisují možnosti a omezení robota. Tvoří základní rozhodovací body taktik. Nalezneme zde, také podrobný popis dovedností. Tyto představují nízko-úrovňové akce, které ovládají pravé, levé kolo a umožňují nám kontrolovat pohyb robota po hřišti.

Klíčová slova: Počáteční inicializace, volba taktiky, vlastnosti robota, globální vlastnosti, dovednosti

Abstract

This diploma thesis is dedicated to control of our robots for robot soccer game. The first phase is focused on initial setting, which is mostly concerned with the computer, the transmitter and the robot. It also includes procedures for setting individual peripherals, to avoid errors due to poor configuration. In addition it has been performed several extensive experiments with robots, which results and evaluation are part of the text. The thesis describes the selection of tactics for individual robots, depending on real data from the field. This data are processed by features, describing the possibilities and limitations of the robot. Creates basic decision points in tactics. Detailed description of these skills can be found here. These skills are low-level actions that control right, left wheel and allow us to control the movement of the robot on the playing field.

Keywords: Initial setting, choice of tactics, robot features, global features, skills

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

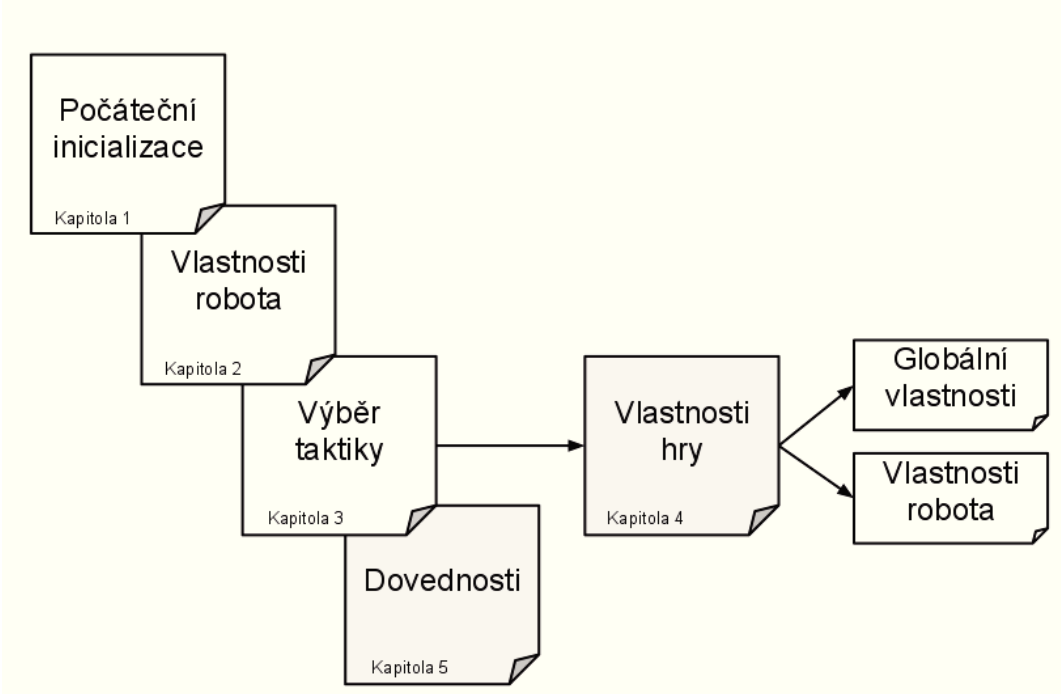
Hra fotbal robotů představuje velmi populární a oblíbenou vědeckou činnost. Toto byl také důvod, čím mě tento projekt zaujal a rozhodl jsem se v rámci diplomové práce, být jeho součástí. Práce nabízí mnoho vědeckých disciplín, které byly rozděleny mezi několik studentů. Teprve až celek reprezentuje reálnou hru fotbal robotů.

Hra je ještě zajímavější tím, že existuje několik komunit, jako je FIRA [5]. Ty stanovily přesná pravidla a pořádají turnaje mezi univerzitami. Mezi nimi vznikla rivalita a jsou následně inspirovány ke zdokonalování svých aplikací. Snahou je tedy dosáhnout co možná nejlepších výsledků v jednotlivých zápasech a samozřejmě pokusit se vyhrát celý turnaj.

Logika hry je rozdělena do dvou úrovní: strategie a taktiky. Cílem strategie je zajistit optimální rozvržení robotů na hrací ploše. Tedy tak, aby byl pokryt prostor před bránou v době, kdy míč kontroluje soupeř. Naopak máme-li míč my, je potřeba obsadit pozice, které povedou ke střelení gólu. Taktiky oproti strategiím nevybírají pozice, ale akce jak těchto pozic dosáhnout. Každá situace si bude žádat volby jiné akce. Taktika zabezpečuje výběr té nejideálnější dovednosti pro danou situaci.

V rámci týmu studentů vzniklém na katedře informatiky jsem se zaměřil především na komunikační část a taktiky. Komunikace s roboty již existovala, ale jednalo se o software, který byl dodán k pořízeným robotům. Cílem tedy bylo vytvořit naší vlastní třídu pro komunikaci s roboty, která bude zcela nezávislá. Existují dva způsoby jak hrát fotbal robotů. Jedná se o hru s reálnými roboty, kdy se pohybují na opravdovém hřišti nebo o simulovanou hru, kde celé prostředí je vytvořeno aplikací. Naše taktiky budou navrženy především pro hru s reálnými roboty. Nic by, ale nemělo bránit v jejich použití pro simulátor. Jen bude potřeba upravit konstanty pro simulované roboty.

Práce bude rozdělena do šesti kapitol, které jsou znázorněny na obrázku 1. Kapitola 1 pojednává o konfiguraci jednotlivých periférií, s kterými jsem pracoval. Jedná se o velmi důležitou část práce, protože nenastavíme-li všechny části dobře, nebudou spolu komunikovat. K robotům nebyl k dispozici žádný manuál, popisující jejich schopnosti a dovednosti, proto v kapitole 2 budou prezentovány prováděné testy na robotovi. Ty byly zaměřeny na získání co možná nejvíce informací o vlastnostech robota. Součástí každého experimentu je vždy postup a také tabulka, jenž představuje výsledek. Tato kapitola bude ukončena zhodnocením všech výsledku a poznatků, které z nich vyplynuly. Užitečnost těchto informací se prokázala u mnoha výpočtů. V kapitole 3 budou popsány jednotlivé vrstvy, jež ovlivňují volbu akce robota. Neboli taktiky. V kapitole 4 dojde k podrobnému rozebráni všech vlastnosti, jak globálních tak robota, které budou pro názornou ukázku vyobrazeny na obrázcích. Ke každé vlastnosti bude podrobně rozepsán algoritmus, který reprezentuje její výpočet. Čtenář je pak schopen lépe a rychleji pochopit problematiku jednotlivých vlastností. V kapitole 5 se zaměříme na samotné dovednosti, které představují primární část pohybového systému. Ty jsou zodpovědné za veškerý pohyb na hřišti. I zde se budu snažit podrobně rozepsat algoritmy, tak aby čtenář byl rychle zasvěcen do problému.



Obrázek 1: Schéma diplomové práce

2 Počáteční inicializace

Ke hře fotbalu robotů je potřeba několik zařízení, která jsou bez manuálu složitá na pochopení. Bez nich, ale nejsme schopni hru realizovat. Obrázek 2 představuje komunikaci kamery, počítače, vysílače a robotů. Tyto části na sobě závisí. Před započetím hry, musíme jednotlivé zařízení nakonfigurovat, kvůli vzájemné komunikaci. Dojde-li k chybnému nastavení, byt'jediné části, roboti nebudou reagovat. Během práce na diplomové práci, jsem se především zabýval komunikací mezi počítačem - vysílačem a vysílačem - robotem. V následujícím textu podrobně popíšu nastavení každého zařízení. Informace ohledně propojení periférií byly čerpány z [8].

2.1 Počítač - vysílač

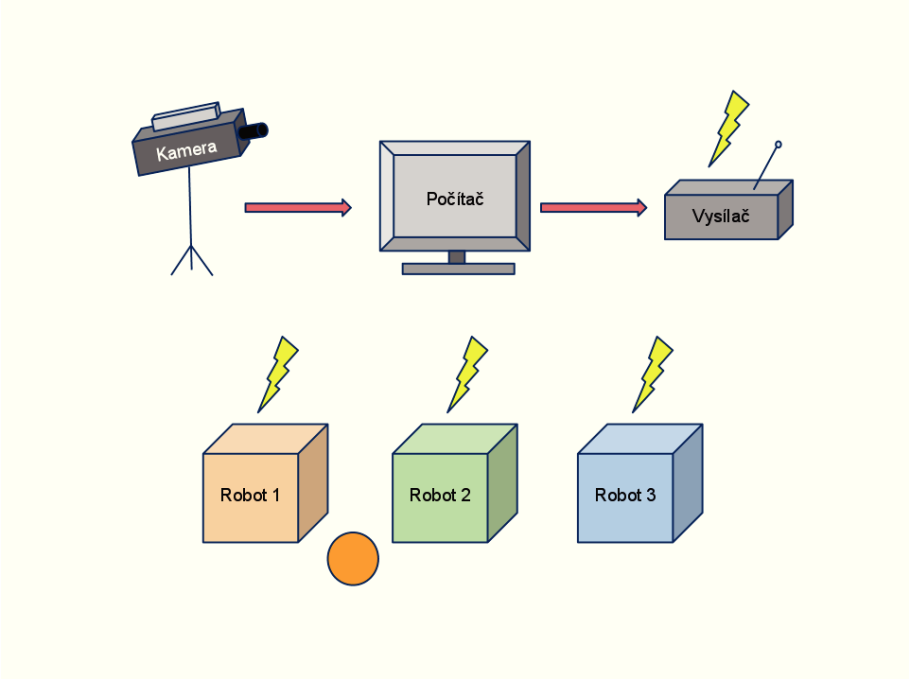
V počítači probíhají veškeré výpočty, které se projeví nastavením rychlostí kol robotům, abychom byli schopni řídit pohyb robota na hřišti. Získané informace je potřeba předat každému robotovi. Komunikaci mezi počítačem a vysílačem probíhá prostřednictvím COM kabelu. Z toho je jasně patrné, že zde se bude jednat především o konfiguraci COM zařízení. Mým úkolem bylo vytvořit třídu pro komunikaci s vysílačem. K vysílači nebyl žádný popis, jak provádět komunikaci a posílat data. Avšak existoval program, který byl součástí vysílače a robotů *serial_test*. Smyslem programu bylo otestovat správnost nastavení komunikace mezi počítačem - vysílačem a robotem. V jednoduchém grafickém rozhraní uživatel zadával rychlosti kol robotům a následně tyto informace pomocí vysílače zaslal. Cílem tedy bylo ze zmíněného programu získat potřebnou komunikaci. Následně vznikla třída *Comm*, která provádí veškeré nastavení COM portu. Během vytvoření instance třídy, se v konstruktoru (výpis 1) nastaví všechny parametry COM portu.

```
Comm::Comm()
{
   idComDev = NULL;
   fConnected = FALSE;
   this->SetComPort(5, 19200, 8, 0, 0);
   this->CreateCommInfo();
   this->OpenComPort();
}
```

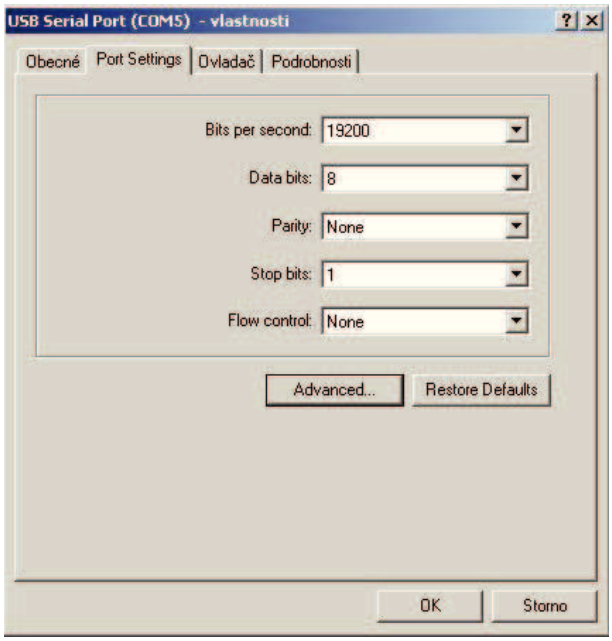
Výpis 1: Konstruktor třídy Comm

Metoda „SetComPort“ (výpis 2) obsahuje parametry, které se nastaví COM portu tj. sériový port, přenosová rychlost v bitech, datové bity, stop bit a paritní bit. Kromě sériového portu je nutno nastavit v počítači COM port podle obrázku 3.

```
void Comm::SetComPort(int port, DWORD rate, BYTE bytesize, BYTE stop, BYTE parity)
{
   bPort = port;
   dwBaudRate = rate;
   bByteSize = bytesize;
   bStopBits = stop;
   bParity = parity;
}
```



Obrázek 2: Komunikace periférií [8]



Obrázek 3: Nastavení COM portu

Výpis 2: Metoda SetComPort ze třídy Comm

Naopak sériový port se v programu musí upravit podle COM portu v počítači tj. v našem případě sériový port je nastaven na hodnotu 5. Chceme-li nastavit data která se mají poslat robotům, zavoláme metodu $VehicleSpeed$ (výpis 3). Metoda má jeden parametr a to pole s rychlostmi všech robotů. Dále metoda je vytvořena tak, aby se dynamicky přizpůsobila změně počtu robotů. Stačí jen upravit konstantu $NUMBER\_OF\_ROBOTS$, ve třídě *GameSetting*, která určuje počet robotů na hřišti. V metodě dochází k naplnění pole $command$ rychlostmi robotů.

```
void Comm::VehicleSpeed(int* pole)
{
    int sizeOfArray = gs->NUMBER_OF_ROBOTS()*2+2;
    unsigned char* command = new unsigned char [sizeOfArray];
    int L,R;
    int x=0;
    int y=0;
    command[x] = 0x00;
    x++;
    for(x;x<sizeOfArray-2;x++)
    {
        L = pole[y] +127;
        R = pole[y] +127;
        command[x] = (unsigned char)R;
        x++;
        command[x] = (unsigned char)L;
        y++;
    }
    command[x] = 0x00;

    WriteCommBlock ((LPSTR)command, 8);
}
```

Výpis 3: Metoda VehicleSpeed ze třídy Comm

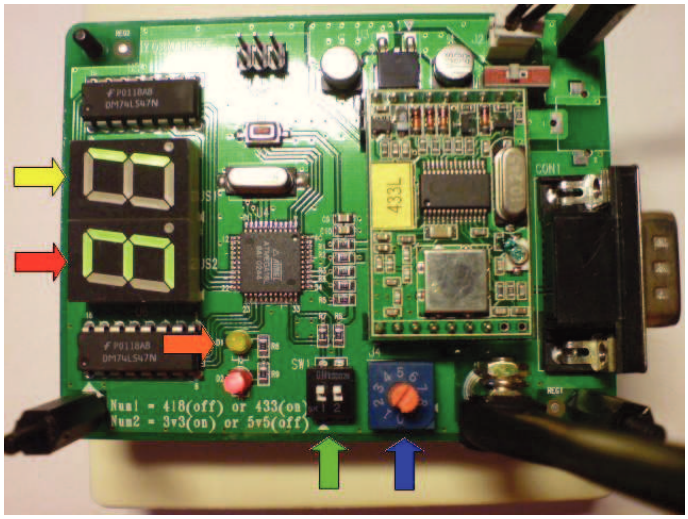
Pole $command$ se dále předá jako parametr metodě $WriteCommBlock$ (výpis 4). V ní dochází k zaslání dat prostřednictvím COM portu.

```
BOOL Comm::WriteCommBlock( LPSTR lpByte , DWORD dwBytesToWrite)
{
    BOOL        fWriteStat  ;
    DWORD     dwBytesWritten ;

    fWriteStat  = WriteFile( idComDev, lpByte, dwBytesToWrite,
        &dwBytesWritten, &osWrite ) ;

    return ( TRUE ) ;
}
```

Výpis 4: Metoda WriteCommBlock ze třídy Comm.cpp



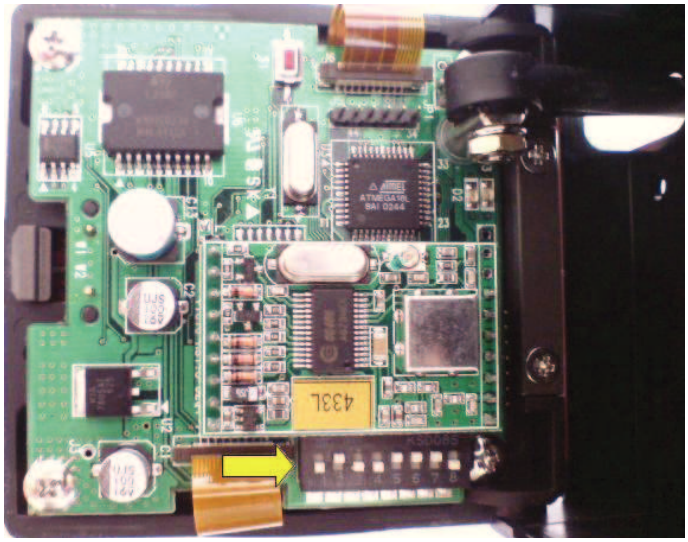
Obrázek 4: Vysílač k ovládání robotů

2.2 Vysílač - robot

Zaslaná data z počítače na vysílač je potřeba dále zprostředkovat každému robotovi na hřišti. Robot s vysílačem nemůže komunikovat skrze datový kabel, jelikož ten by do značné míry omezoval pohyb robota. Komunikace mezi vysílačem a robotem je postavena na bezdrátové rádiové komunikaci. Vysílač má následující parametry:

1. vysílací frekvence - 418 MHz nebo 433 MHz,
2. polo duplexní přenos dat,
3. počet nastavitelných kanálů - 10,
4. přenosová rychlost - 19200 bps,
5. přijímající citlivost - -107 dBm.

Na obrázku 4 vidíme náš vysílač a šipky znázorňují nejdůležitější součástky pro jeho konfiguraci. Jak již bylo zmíněno v základním popisu, vysílač obsahuje přepínač mezi deseti kanály. Na tento přepínač poukazuje modrá šipka. Jeho otáčením si navolíme požadovaný kanál. Hned po jeho levici se nachází dvou DIP-ový přepínač, je znázorněn zelenou šipkou. První DIP určuje frekvenci vysílače, na které se bude vysílat data robotům. Můžeme si vybrat mezi dvěma frekvencemi 418 MHz při nastaveni DIP-u na OFF nebo 433 MHz přesuneme-li DIP do pozice ON. Druhý DIP slouží k určení počtu ovládaných robotů. Opět můžeme volit mezi dvěma nastaveními. Je-li DIP v pozici OFF nastavíme vysílač pro hru pět na pět, v opačném případě tj. ON nastavíme vysílač pro hru tři na tři. Vysílač má zabudovaný dva sedmi segmentové displeje. První znázorněn žlutou šipkou, informuje o zvoleném kanálu vysílače. Druhý znázorněn červenou šipkou, informuje o zvolené frekvenci pro komunikaci. Displej zobrazuje poslední číslo frekvence, tj. v



Obrázek 5: Konfigurace robota

	$DIP\ SW\ \#1$	$DIP\ SW\ \#2$	$DIP\ SW\ \#3$
$robot\ \#1$	OFF	ON	ON
$robot\ \#2$	ON	OFF	ON
$robot\ \#3$	ON	ON	OFF

Tabulka 1: Nastavitelná ID pro roboty [8]

případě 418 MHz zobrazuje se číslice 8, naopak pro 433 MHz zobrazuje 3. Zůstává už jen poslední nezmíněná věc, kterou znázorňuje oranžová šipka a tj. zelená LED dioda. Její význam spočívá v indikaci přenosu dat. Dochází-li k posílání dat a roboti nereagují, pak jako první je potřeba zkontrolovat právě zmíněnou diodu. Každé posílání dat mění stav diody, tj. rozsvítí a zhasne. Takto můžeme na první pohled identifikovat, kde se nachází chyba v nastavení. Nedochází-li k blikání diody jedná se nejčastěji o špatné nastaveni komunikace mezi počítačem a vysílačem. Jakákoli změna v nastavení se projeví až po restartování vysílače. Upravíme-li tedy například kanál nebo frekvenci, je nutno vysílač odpojit ze sítě a opětovně zapojit. Neprovedeme-li restart, vysílač se chová tak jak byl nastaven než došlo k úpravě. Nastavení vysílače je pouze první části. Roboty je potřeba také nastavit podle zvolených parametrů na vysílači, aby byli schopni přijímat zasílaná data. Celé nastavení robota se provádí na osmi DIP-ovém přepínači, který je zobrazen na obrázku 5. Chceme-li ovládat roboty musíme nastavit jednotlivým robotům jejich ID, abychom byli schopni je rozlišit. Podle nastaveného ID robot rozpozná, které informace jsou určeny právě jemu. Pomocí prvních tří DIP-ů určíme ID robota. Tabulka 1 určuje nastavení ID pro tři roboty. Pomoci DIP-u 4 až 7 nastavíme kanál podle vysílače. Tabulka 2 představuje seznam nastavení DIP-ů pro jednotlivé kanály. Zbývá poslední osmý DIP, který slouží k nastavení frekvence podle vysílače. Je-li DIP v poloze ON pak robot má nastavenou frekvenci 418 MHz, v případě OFF se jedná o frekvenci 433 MHz.

	$DIP\ SW\ \#4$	$DIP\ SW\ \#5$	$DIP\ SW\ \#6$	$DIP\ SW\ \#7$
$Channel\ 1$	OFF	ON	ON	ON
$Channel\ 2$	ON	OFF	ON	ON
$Channel\ 3$	OFF	OFF	ON	ON
$Channel\ 4$	ON	ON	OFF	ON
$Channel\ 5$	OFF	ON	OFF	ON
$Channel\ 6$	ON	OFF	OFF	ON
$Channel\ 7$	OFF	OFF	OFF	ON
$Channel\ 8$	ON	ON	ON	OFF
$Channel\ 9$	OFF	ON	ON	OFF
$Channel\ 10$	ON	OFF	OFF	OFF

Tabulka 2: Nastavení kanálů pro roboty [8]

Při nastavování robotů musíme nastavit pro všechny stejný kanál a také frekvenci, na které dochází ke komunikaci s vysílačem. V případě špatného nastavení roboti nebudou přijímat data. Nastavení robotů se liší pouze v jejich ID, které je unikátně identifikuje. Díky tomu můžeme každému robotu nastavit jiné rychlosti kol. Tak jako u vysílače upravíme-li nastavení robota, musíme provést restart tím, že ho vypneme a opětovně zapneme. Bez restartu se změny nastavení neprojeví. Během všech testů byl vysílač nastaven s těmito parametry:

1. frekvence - 433 MHz,
2. kanál - 1,
3. počet robotů - 3 vs 3.

3 Vlastnosti robota

Existuje mnoho odvětví fotbalu robotů a každá z nich je založená na odlišné konstrukci robota. Chceme-li maximálně využít možnosti robota, musíme nejdříve dopodrobna prozkoumat a pochopit jeho vlastnosti a schopnosti. Bez těchto základů nejsme schopni připravit algoritmy, které nám zajistí pohyb robota na hřišti podle našich představ. První věcí, kterou ihned zpozorujeme během testování robota, je jeho tvar a rozměr. Ač se to na první pohled může zdát jako naprosto nepotřebná informace, opak je pravdou. Během hry dochází k mnoha výpočtům, kde je potřeba znát šířku robota. Počítáme-li vzdálenosti mezi robotem a cílovým objektem, obdržíme vždy délku ke středu robota. Je to způsobené analýzou, jejímž cílem je detekce robota a následně určit jeho souřadnice na hřišti. Ty jsou počítány jako střed robota. Potřebujeme-li tedy znát skutečnou vzdálenost mezi objekty, jsme nuceni počítat s jeho šířkou. Náš robot se podobá kostce, s rozměry 7 cm na výšku, 7,5 cm na délku a šířku. Robot je prezentován na obrázku 6. Jako další podstatnou vlastností robota je jeho pohybová schopnost a především mechanizmus, který pohyb zprostředkovává. Tedy pro náš model je na první pohled patrné, že pohyb je zajištěn dvěma postranními koly. Pouze s postranními koly si však robot nevystačí, docházelo by k neustálému naklánění dopředu a dozadu. To by mělo za následek sníženi pohybové rychlosti po hřišti, v důsledku nadměrného tření o hrací plochu. Především třením by robot mohl být vychylován z požadovaného kurzu. Tento problém je řešen malými přídavnými kolečky ve předu a vzadu. Ty spolehlivě zajišťují, aby se robot nacházel neustále ve stabilizované poloze. Bavíme-li se o kolech a pohybovém mechanizmu, je potřeba popsat, jak vlastně celý ten proces uvnitř funguje. Systém robota je rozdělen do čtyř částí, které spolu vzájemně komunikují, tj. mikrořadič, řízení motorů, komunikace a napájení. Porucha jedné částí bude mít za následek nefunkčnost robota, z toho nám jasně vyplývá, že části jsou na sobě závislé. Nyní si trochu řekneme o vlastnostech jednotlivých části:

1. komunikace - přijímá komunikační příkazy od počítače, který řídí celou hru,
2. napájení - zajišťuje energii pro provoz motorů a mikrořadiče,
3. mikrořadič - překládá komunikační příkazy od počítače a vytváří řídící příkazy pro motory,
4. řízení motorů - poskytuje elektřinu motorům, podle informací, které obdržel z řídícího příkazu pro jednotlivé motory.

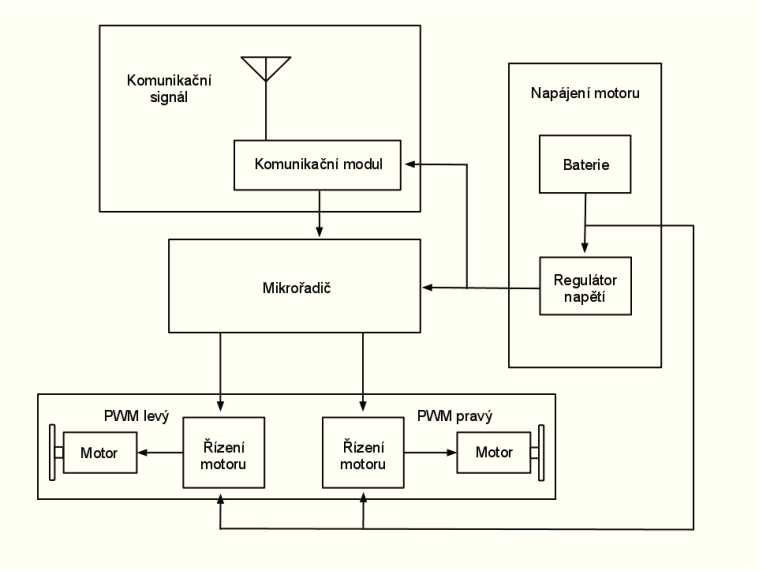
Celý systém pohybového mechanizmu robota je zobrazen na obrázku 7. Napětí nabité baterie se pohybuje kolem 7,4V a je využíváno jako energie pro motory. Napětí 7,4V sestupuje až na 5V skrze regulátor napětí. 5V je využito pro logický obvod. Princip pohybu robota je založen na PWM, tzv. pulzní šířkové modulaci [8]. Jedná se o metodu používanou k řízení podílu mezi vysokým a nízkým napětím, během jedné periody vzorkovacího času. „Duty rate“ je poměr mezi vysokým a nízkým napětím, díky čemu můžeme ovlivnit směr otáčení motoru. Nejvýznamnější použití PWM metody, je vyjádření pracovních cyklů, jako hodnoty v rozsahu 0 až 255. Díky přenášenému napětí do motoru pomocí PWM, jsme schopni kontrolovat směr otáčení motoru. Pásmo 0 až 255 představuje oblast, ve které

Obrázek 6: Robot pro hru Fotbal robotů

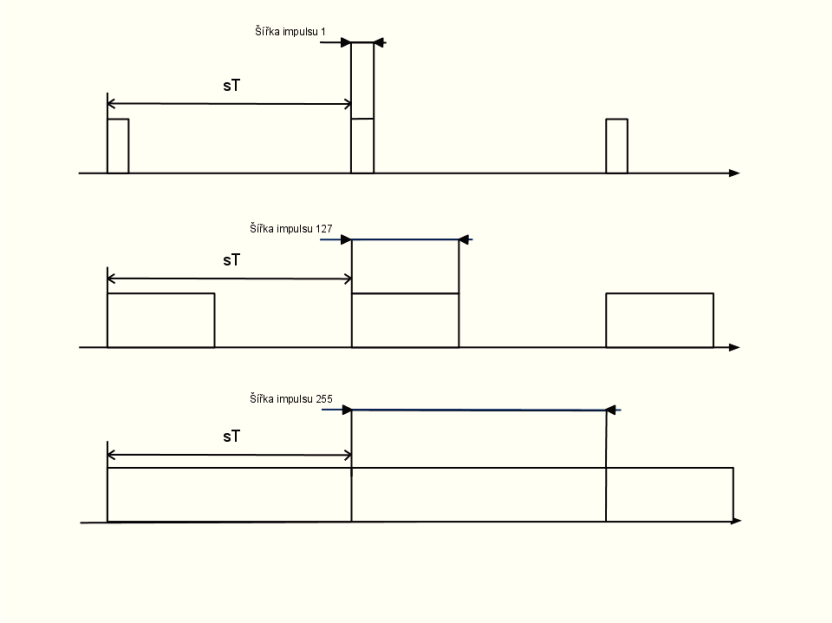
máme možnost určovat rychlost každému kolu robota. Na obrázku 8 je znázorněn princip pulzní šířkové modulace. Centrální hodnota, která reprezentuje číslo 0, tj. rychlost, kdy se kolo robota neotáčí, je 127. V programu, musíme ke každé rychlosti kola tuto hodnotu připočítat, abychom dosáhli námi požadované rychlosti a směru. Naopak, má-li robot jet pozpátku, hodnoty se od tohoto čísla budou odečítat. Z toho vyplývá, že hodnoty nad 127 reprezentují zvětšující se rychlost dopředu a naopak hodnoty pod 127 zvětšující se rychlost dozadu. Robot není nijak omezen směrem, jeho výkon je identický na obě strany a toho můžeme při hře dobře využít. Informace o systému robota byly čerpány z [8].

3.1 Testy vlastností

O robotovi máme k dispozici hodně textu [8], [2], [1], kde se dočteme na jakém principu robot funguje, jak probíhá komunikace, zda se jedná o rádiový přenos, či přenos pomocí infrazáření. Jak nastavit rychlosti a v jakém rozsahu hodnot se můžeme pohybovat. Co se ovšem nedovím jsou například skutečné rychlosti při jednotlivých hodnotách nastavených robotovi. Jaké maximální rychlosti je robot schopen dosáhnout nebo nejmenší nastavitelnou rychlost, během které se robot opravdu pohybuje. Samozřejmě nebudeme vyžadovat jenom pohyb ve dvou směrech, ale od robota se bude očekávat také zatáčení a otáčení na místě. Zde jsou na místě otázky za jak dlouho se robot dokáže otočit, nejmenší poloměr otáčení s největší možnou rychlostí. Na tyto a další otázky bylo snahou připravit testy, které by nám daly odpovědi. Veškerá měření byla prováděna v omezených prostorách a časy zaznamenány na analogových stopkách. K provedení testů nebyly prostředky, abych mohl získat přesná data. K tomu by bylo zapotřebí snímačů, které by přesně měřily



Obrázek 7: Základní části systému robota [8]



Obrázek 8: Pulzní šířková modulace [8]

Rychlost	Délka[m]	Čas[s]	Reálná rychlost[m/s]	Reálná rychlost[km/h]
5	2	25,5	0,0784	0,2822
25	2	5,2	0,3846	1,3846
50	2	2,6	0,7692	2,7691
75	2	1,8	1,1111	4
100	2	1,4	1,4286	5,143
125	2	1,2	1,6667	6,0001

Tabulka 3: Výsledek testu rychlosti

start a konec pohybu robota. To může mít za následek, že se naměřené údaje budou od skutečných trochu lišit. Myslím, že pro naší snahu popsat následující vlastnosti robota to nebude mít příliš velký dopad, jelikož snahou je, aby uživatel získal představu o jeho schopnostech. Nyní se zaměříme na prováděné testy a informace, které jsme na základě nich získali. Pro testovací účely vznikla aplikace *Robots*, která byla použita u všech testů. S její pomocí jsme mohli nastavovat rychlosti robotům a následně je podrobit jednotlivým testům.

3.1.1 Test rychlosti

Cílem tohoto testu, je získaní informací o reálných rychlostech, pro jednotlivé rozsahy.
Popis: na podlaze jsme vyznačili vzdálenost 2m. Takto vznikly dvě rovnoběžné přímky, které vyznačovaly start a konec trasy. Robota jsme umístili na začátku. V programu pro ovládání robota, ve třídě *Comm* došlo k nastavení identické rychlostí pro levé a pravé kolo. Spustil se program. V okamžiku, kdy se robot dal do pohybu, sepnul jsem stopky a jakmile překonal vzdálenost 2m došlo k jejich zastavení. Takto jsme získali cennou informaci, za jak dlouho robot překonal vzdálenost 2m, z čehož následně podle vzorce:

$$v = \frac{s}{t},$$

vypočítáme průměrnou rychlost. Výsledek měření nalezneme v tabulce 3.

3.1.2 Test brzdné dráhy

Cílem tohoto testu, je získaní informace o brzdné dráze robota, pro jednotlivé rozsahy rychlostí. Z testu rychlostí jsme získali informaci, za jak dlouho dojde k překonání vzdálenosti 2m. Tento test je na tomto údaji závislý. Během testu využíváme metodu $Sleep()$, k pozastavení programu.
Popis: Robota opět umístíme na začátek. Ve třídě *Comm* nastavíme identické rychlosti pro levé a pravé kolo. Dále v metodě $Sleep()$ definujeme čas na jak dlouho se má program pozastavit, než dojde k zaslání rychlosti 0 a tím zastavení robota. Pro jednotlivé rychlosti robota nastavujeme, jím odpovídající časy z tabulky 3. Robot po dobu času se pohybuje kupředu s konstantní rychlosti. Jakmile uběhne námi definovaný čas, dojde k zaslání příkazu pro zastavení. Vzdálenost robota za cílovou čárou představuje brzdnou dráhu. Tuto

Nastavení rychlosti	Délka dráhy[m]	Čas[s]	Brzdná dráha[cm]
5	2	25, 5	0
25	2	5, 2	0
50	2	2, 6	1
75	2	1, 8	10
100	2	1, 4	20
125	2	1, 2	58

Tabulka 4: Výsledek testu brzdné dráhy

Nastavení rychlosti	Délka dráhy[m]	Čas[s]
5	2	25, 5
60	2	2, 2
125	2	0, 85

Tabulka 5: Výsledek testu zastavení na místo

vzdálenost změříme a zaznamenáme do tabulky. Výsledek měření nalezneme v tabulce 4.

3.1.3 Test zastavení na místo

Cílem testu bylo zjistit, pro jednotlivé rychlosti čas, kdy robot s brzdnou dráhou zastaví přesně na cílové čáře. Tak aby zastavil na bodě ve vzdálenosti 2m. Pro definování časového intervalu použijeme opět metodu $Sleep()$.
Popis: Robota postavíme na startovní čáru. V programu ve třídě *Comm* nastavíme rychlost, pro kterou chceme zjistit čas potřebný, k přesnému dosažení cíle. V metodě $Sleep()$ nastavíme časový interval, během kterého se robot bude pohybovat a po jeho uplynutí dojde k zastavení. Po spuštění robot jede kupředu s konstantní rychlostí a po námi definovaném časovém intervalu se zastaví. Podle pozice robota upravíme časový interval, tj. je-li před cílovou čárou je potřeba zvětšit časový interval respektive zmenšit je-li za cílovou čárou. Jakmile se zastaví přesně na čáře, zjistili jsme interval pro námi testovanou rychlost. Výsledek měření nalezneme v tabulce 5.

3.1.4 Test otočení o 360°

Cílem testu je získat přehled, za jak dlouho, se při jednotlivých rychlostech, dokáže robot otočit o 360° na místě. Pro měření použijeme metodu $Sleep()$.
Popis: Na podlaze se nakreslily dvě kolmé přímky. Tím vznikl kříž, v jehož středu jsme umístili robota. V programu ve třídě *Comm* nastavíme identickou rychlost robotovi, pro obě kola, s tím rozdílem, že jedno kolo tuto hodnotu bude mít kladnou a druhé zápornou. Takto robot dosahuje nejrychlejšího otočení na místě. Pomocí metody $Sleep()$ se budeme snažit, pro danou rychlost najít čas, během kterého se robot otočí na místě do původní pozice. Otočí-li se robot o více stupňů, pak je potřeba čas zmenšit. Otočí-li se robot o méně

Nastavení rychlostí	Čas[s]
5	2, 45
20	0, 605
25	0, 485
50	0, 25

Tabulka 6: Výsledek testu otočení o 360°

Levé kolo	Pravé kolo	Rozdíl
125	120	5
110	103	7
100	91	9
90	80	10
80	68	12
70	55	15
60	31	19
50	18	32
40	5	35

Tabulka 7: Výsledek testu maximálních rozdílu mezi koly

stupňů, pak je potřeba čas zvětšit. Jakmile robot dosáhne původní pozice, získali jsme časový interval otočení, pro určitou rychlost. Výsledek měření nalezneme v tabulce 6.

3.1.5 Test maximálních rozdílu mezi koly

Cílem testu je získání maximálních rozdílů v rychlostech mezi koly, kdy robot ještě jezdí v pravidelných kruzích.
Popis: Robota jsme umístili uprostřed místnosti, aby měl dostatek místa k manévrování. V programu pro ovládání robota ve třídě *Comm* nastavíme vždy na levé kolo konstantní rychlost. Rychlost na pravém budeme postupně od rychlosti levého zmenšovat a program vždy spustíme. Pohybuje-li se robot neustále v pravidelném kruhu, provedeme opětovné zmenšení rychlosti na pravém kole. Začne-li se robot nekontrolovatelně pohybovat, pak předešlá hodnota je maximální nejnižší rozdílná rychlost vzhledem k rychlosti na levém kole. Výsledek měření nalezneme v tabulce 7.

3.2 Testy pohybu

Po vytvoření třídy *Comm*, která slouží k ovládání pohybu robotů, bylo potřeba demonstrovat pohybovost a ovladatelnost několika robotů. Za tímto účelem vznikly tři testovací metody: čtverec, kruh, nekonečno. Cílem metod bylo otestování ovladatelností několika robotů současně. V průběhu vývoje našly ještě jedno uplatnění. Jakmile vývoj analýzy obrazu byl v takovém stádiu, kdy bylo možné pořídit video sekvence, na těchto metodách se provedly záznamy. Experiment nesloužil pouze k otestování analýzy, ale především

získané video sekvence posloužili dalším studentům, jako materiál, na kterém mohli provádět svoje experimenty, především práce na filtrech obrazu. V metodách nebylo možné počítat s údaji o pozici robotů na hrací ploše, protože analýza obrazu [3] ještě nebyla schopna detekce objektů. Metody jsou založeny na časových intervalech, pro jejich definování se použila metoda $Sleep()$.

3.2.1 Metoda čtverec

Metoda (výpis 5) má demonstrovat pohyb robota pohybujícího se do čtverce. Ze získaných hodnot o zatáčení robota, pro jednotlivé rychlosti si můžeme spočítat, za jak dlouho je schopen se otočit o 90 stupňů. Během zatáčení je nutno metodě $Sleep()$ tento časový interval nastavit, jenž určuje, jak dlouho má robot provádět otáčení. Není vyloučeno, že čas nebude odpovídat každému prostředí, jelikož zde velmi záleží na povrchu. Proto je potřeba toto číslo trochu přizpůsobit. Druhým časovým intervalem ovlivníme velikost čtverce. Tedy dobu trvání jednoho cyklu.
Postup: Metoda začíná nastavením počátečních rychlostí robotům. Cyklus $while$ ovlivňuje počet zatočení, které roboti mají vykonat. První $Sleep()$ určuje dobu jednoho cyklu. Pak následuje nastavení větší rychlosti levého kola. Takto robot provede plynulý obrat. Otáčení je definováno druhou metodou $Sleep()$. Jakmile zatočení skončí, nastavíme původní rychlost pro levé kolo. Na konci dochází k zastavení všech robotů.

```
void Tests::Ctverec()
{
    int pocetZatoceni = 0;
    int ukazatel = 0;
    for(int i=0;i<count;i++)
    {
        robotsArray[i]->SetRobotL(20);
        robotsArray[i]->SetRobotR(20);
    }
    comm->VehicleSpeed(robotsArray);
    while(pocetZatoceni != 5)
    {
        Sleep(1500);
        for(int i=0;i<count;i++)
        {
            robotsArray[i]->SetRobotL(31);
        }
        comm->VehicleSpeed(robotsArray);;
        Sleep(565);
        for(int i=0;i<count;i++)
        {
            robotsArray[i]->SetRobotL(20);
        }
        comm->VehicleSpeed(robotsArray);
        pocetZatoceni++;
    }
    for(int i=0;i<count;i++)
    {
        robotsArray[i]->SetRobotL(0);
```

```
        robotsArray[i]->SetRobotR(0);
    }
    Sleep(200);
    comm->VehicleSpeed(robotsArray);
}
```

Výpis 5: Metoda Ctverec ze třídy Tests

3.2.2 Metoda kruh

Metoda (výpis 6) demonstruje pohyb robota do kružnice. Oproti metodě čtverec zde není potřeba řešit žádné časové intervaly. Při pohybu do kružnice nedochází ke změně směru a tudíž není potřeba modifikovat rychlosti. Pouze místo cyklu, který předtím určoval počet zatočení, použijeme metodu $Sleep()$ k definování doby provádění kružnice.
Postup: Metoda začíná nastavením počátečních rychlostí robotům. Rychlosti jednotlivých robotů se liší a to tak, že každý robot má jinou hodnotu pro rychlost pravého kola. Tím je zajištěno, aby každý robot vykonával trochu odlišnou kružnici. V metodě $Sleep()$ definujeme délku provádění testu. Na konec dochází k zastavení všech robotů.

```
void Tests::Kruh()
{
    int r = 15;
    for(int i=0;i<count;i++)
    {
        robotsArray[i]->SetRobotL(r);
        robotsArray[i]->SetRobotR(r+10);
        r += 10;
    }
    comm->VehicleSpeed(robotsArray);
    for(int i=0;i<count;i++)
    {
        robotsArray[i]->SetRobotL(0);
        robotsArray[i]->SetRobotR(0);
    }
    Sleep(2000);
    comm->VehicleSpeed(robotsArray);
}
```

Výpis 6: Metoda Kruh ze třídy Tests

3.2.3 Metoda nekonečno

Metoda (výpis 7) demonstruje pohyb robota do osmičky. Cílem je demonstrovat schopnost robota plynule modifikovat směr a změnu rychlosti na kolech. Abychom mohli realizovat tuto metodu potřebujeme znát čas, za jak dlouho robot objede kruh, pro určitou rychlost kol. Vždy když uběhne tento časový interval, robot dokončil kruh a je potřeba nastavit nové rychlosti. Tím je dosaženo změny směru a robot vykresluje osmičku.
Popis: robotovi nastavíme rychlosti kol, pro které máme změřený čas objíždění kruhu. Cyklus $while$ představuje počet objetých osmiček. Pomocí metody $Sleep()$ pozastavíme

proces po dobu časového intervalu, aby robot měl čas objet celý kruh. Pak dojde k přehození rychlostí mezi levým a pravým kolem. Robot objede stejně velký kruh na druhou stranu. Po dobu objíždění je opět pozastaven proces. Nyní je dokončena osmička a robot začíná znovu od začátků. Jakmile jsou provedeny všechny, nastavíme robotovi nulové rychlosti, aby zastavil. Tím je demonstrace ukončena.

```
void Tests::Nekonecno()
{
    int smycka = 0;
    for(int i=0;i<count;i++)
    {
        robotsArray[i]->SetRobotL(40);
        robotsArray[i]->SetRobotR(30);
    }
    comm->VehicleSpeed(robotsArray);
    while(smycka != 5)
    {
        Sleep(2550);
        for(int i=0;i<count;i++)
        {
            robotsArray[i]->SetRobotL(30);
            robotsArray[i]->SetRobotR(40);
        }
        comm->VehicleSpeed(robotsArray);
        Sleep(2550);
        for(int i=0;i<count;i++)
        {
            robotsArray[i]->SetRobotL(40);
            robotsArray[i]->SetRobotR(30);
        }
        comm->VehicleSpeed(robotsArray);
        smycka++;
    }
    for(int i=0;i<count;i++)
    {
        robotsArray[i]->SetRobotL(0);
        robotsArray[i]->SetRobotR(0);
    }
    Sleep(200);
    comm->VehicleSpeed(robotsArray);
}
```

Výpis 7: Metoda Nekonecno ze třídy Tests

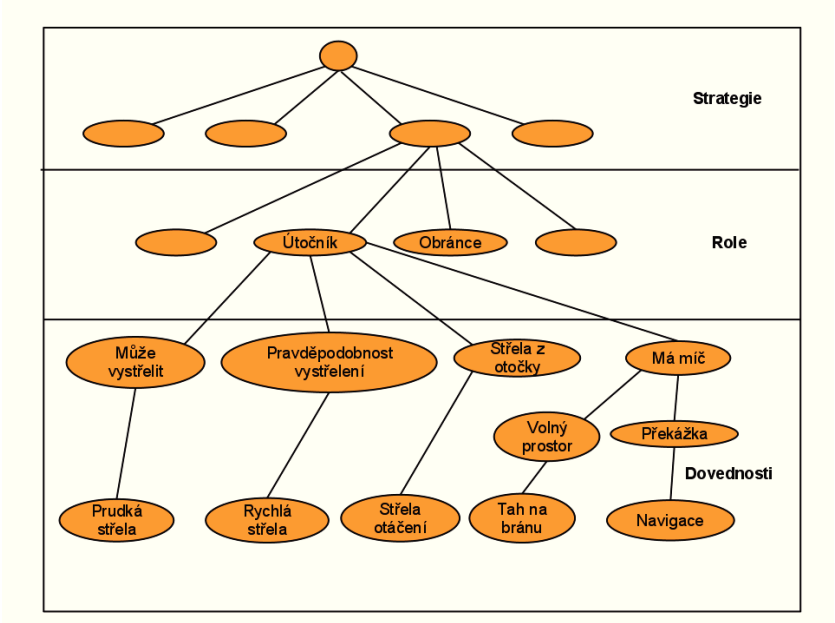
3.3 Shrnutí testů

Pokusím se shrnout nejdůležitější informace, které vyplývají z provedených testů a jiné poznatky, které jsem zaznamenal. Z testů vyplynulo, že nejnižší rychlost, kterou má smysl robotovi nastavit je 5. Při této rychlosti robot vykazuje plynulý pohyb. Pro nižší rychlosti se pohybuje sekavě. Díky tomuto poznatku se v testech s nižšími rychlostmi nepracovalo. Mezi testy byl proveden pokus, zaměřený na schopnost robota přijímat příkazy. Pomocí

metody $Sleep()$ se postupně nastavoval menší časový interval mezi zaslanými příkazy pro pohyb robota a následně pro jeho zastavení. Takže zašleme-li rychle za sebou dva příkazy, musí být mezi nimi minimálně 7ms časový odstup. V opačném případě dochází k vynechávání příkazu. Pravděpodobnost, že by se nám tohle stalo během utkání je velmi nepravděpodobné. Minimálně jenom analýza obrazu [3] si vyžádá více času, než je 7ms. I když konstrukčně je robot navržen velmi dobře, je velmi citlivý na kvalitu plochy, po níž se pohybuje. Vyskytnou-li se na ploše nerovnosti, výrazně se tím ovlivní pohyb robota. Během testování, jak jsem se již zmínil, byly k dispozici omezené prostory s nedokonale rovným povrchem. Robot, pohybující se po této ploše, při nastavené rychlosti od 75, měl tendenci měnit směr jízdy. Abychom docílili úspěšného provedení testu, musela se pro robota sestavit dráha, v které se mohl pohybovat. Takto nedocházelo k odklonění ze směru. Toto opatření, má za následek občasné tření robota a tím dochází k nepřesnému měření. Na testy není kladen vysoký nárok na přesnost, jelikož slouží jako informativní a mají posloužit v budoucím sestavování taktik a základních pohybových dovedností robota. Z testu vyplývá, že nejmenší pohybová rychlost, která se bude nastavovat robotovi je 5, tj. 0,0784 m/s. Maximální rychlost, kterou robot dokáže vyvinout se pohybuje kolem 1,6667 m/s. Robot byl podroben testu zaměřenému na jeho brzdnou dráhu. Z naměřených testů vidíme, nastavíme-li rychlost do 50 robot je schopen zastavit přesně na místě, pro vyšší rychlosti už je nutno počítat s prodloužením dráhy. Brzdná dráha začíná při rychlosti 50, kde činní přibližně 1cm. S rostoucí rychlostí se pochopitelně zvětšuje, až na 0,5m u maximální rychlosti. Budeme-li chtít, aby se robot rychle přemístil na námi definovanou pozici, je potřeba počítat s delší brzdnou dráhou. Během testování otočení na místě, bylo zaznamenáno, že optimální rychlost pro tuto dovednost robota se pohybuje v rozmezí od 20 do 25. S vyššími hodnotami se začíná projevovat odstředivá síla, kdy robot v podstatě jako u brzdné dráhy se pootočí ještě o několik stupňů. Rychlosti nad 50 vykazují příliš velký odklon od cílové pozice. Rychlost otáčení na místě, bude kritickou částí pohybu. Data, která robot obdrží, získává se zpožděním. Z toho vyplývá, otáčí-li se příliš velkou rychlostí, při které není schopen zastavit přesně ve směru nové pozice, je nucen ujet větší délku. Zastavení přesně ve směru nové pozice, umožní okamžitý pohyb vpřed. Robot tedy vykonává přímý lineární pohyb. Nezastaví-li se přesně ve směru musí odchylku překonat během pohybu na cíl. Robot tedy vykonává krouživý pohyb. Test rozdílu rychlosti mezi koly, poskytuje také velmi cenné informace. Ze získaných údajů je jasně patrné, že při vysoké rychlosti robot není schopen manévrovat natož, aby se rychle otočil. Se snižující rychlosti se rozdíl mezi koly zvětšuje. Nejlepšího manévrování dosáhneme při rychlostech do 40, kde rozdíl činí 35. Tím dostáváme možnost nastavit velký rozdíl mezi rychlostmi kol a robot je schopen zatáčet na velmi malém prostoru. V případě, že pro daný rychlostní rozsah přesáhneme maximální možný rozdíl, robot se začne nekontrolovatelně pohybovat.

4 Výběr taktiky

Každému robotovi je potřeba vybrat ve hře, jakou roli bude zastávat a především jakou činnost je schopen provést. O volbu těchto činností se zabývá taktika. Bavíme-li se o taktikách, je potřeba vysvětlit co to vlastně vůbec taktika je. Jedná se tedy o proces, který na základě jistých podnětů, rozhoduje o volbě akce. Zde taktika představuje chování jednoho robota a podnětem je situace na hřišti. Tedy rozmístění objektů. Akce zase představují dovednosti, které generují nízko-úrovňové příkazy, jež nastaví rychlost motoru. O hře fotbalu robotů, můžeme bezpochyby tvrdit, že se jedná o velice rychlou hru. Každé zpoždění má za následek pomalejší reakce robotů. Výběr akce, proto nemůže trvat příliš dlouho. V případě zdlouhavého výběru se mohou vyskytnout situace, kdy prováděná akce, již nebude aktuální. Například robot se bude snažit vystřelit v místě, kde se již míč nemusí nacházet, popřípadě už nebude v dobré střelecké pozici a jiná akce by v tomto momentě byla mnohem efektivnější. Z tohoto důvodu je náš model navržen, jako soubor reaktivního chování. Jinak řečeno jedná se o akci na základě reakce. Takové chování je lépe srozumitelné z obrázku 9 nebo také z článku, z kterého byly informace k taktikám čerpány [2]. Prohlédneme-li si obrázek pozorně neunikne nám, že základem je rozdělení do tří vrstev. Na první úrovní se nachází strategie. Jejich význam je velmi jednoduchý. Mají za cíl určit nejefektivnější rozmístění robotů, tak abychom dosáhli účinného útoku a v případě obrany, efektivního pokrytí. Strategie zároveň přiřazují robotům role, které v rámci týmu vykonávají. Přiřazení probíhá na základě vzdálenosti robota k míči a také na které polovině hřiště se míč pohybuje. Je-li míč u soupeře, nejbližší robot představuje útočníka. Naopak je-li míč na naší polovině, nejbližší robot představuje obránce. Takové přiřazení má své opodstatnění, protože na každé polovině hřiště, je primárním cílem něco jiného. Rozlišujeme čtyři základní role: útočník, obránce, brankář a nakonec nedefinovaný. Zamyslíme-li se nad jednotlivými rolemi, jejich týmový význam je hned jasný. Útočník se snaží míč dostat do branky soupeře, obránce naopak redukuje co možná největší počet střel na naší bránu, brankář blokuje střely v brankovišti, ale pod pojmem nedefinovaný si už těžko něco představíme. Zařazení čtvrté role má svůj význam. Předpokládejme, že na hřišti se pohybuje pět našich robotů. Rozdělíme-li mezi ně pouze role útočník,obránce,brankář, pak některé role se budou objevovat vícekrát. To však povede, ke kolizi našich robotů. Není potřeba, aby dva útočníci současně se hnali za míčem. Vzájemně by si více škodili. Proto vznikla role nedefinovaný. Takto označený robot bude řízen na pozici určenou strategii a taktiky nebudou určovat jeho pohyb. Tím je dosaženo efektivního rozmístění robotů na hřišti. Druhou úrovní jsou již zmiňované role. Rozdělení podle rolí, je velmi důležité. Každá role sebou nese jiné taktiky a dovednosti robota. Cílem je odseparovat dovednosti, které jsou užitečné jen pro příslušné role. Například dovednost objíždění překážek, je pro brankáře holým nesmyslem. Třetí a poslední úrovní jsou základní dovednosti robota. Každá dovednost představuje unikátní výpočet. Výpočet probíhá nad daty objektů na hřišti. Výsledkem výpočtu, je nastavení efektivních rychlostí levého a pravého kola robota, pro danou dovednost. Vyšší uzly stromu představují rozhodovací body, které jsou rozděleny do dvou skupin: globální vlastnosti a vlastnosti robota. Na základě jejich výsledků se volí akce robotovi. Rozhodovací body jsou testovány podle jejich priority. Priority jsou různé podle polohy míče na hřišti.To



Obrázek 9: Výběr taktiky [2]

znamená, že pro útočníka, v momentě když míč se nachází na soupeřově straně, jako první zjišťuje možnost střely na opravdové souřadnice míče. Další v pořadí pak budou možnosti střely na predikovanou pozici míče. Posledním rozhodovacím bodem pro tuto situaci představuje pohyb za míčem, aby jej získal. Naopak nachází-li se míč na naší polovině, útočník je přemístěn na pozici určenou strategií. Taktiky pro jednotlivé role se nacházejí ve třídě *TacticChooser*. Metodu s taktikami reprezentuje výpis 12. V následující kapitole rozebereme jednotlivé vlastnosti.

5 Vlastnosti hry

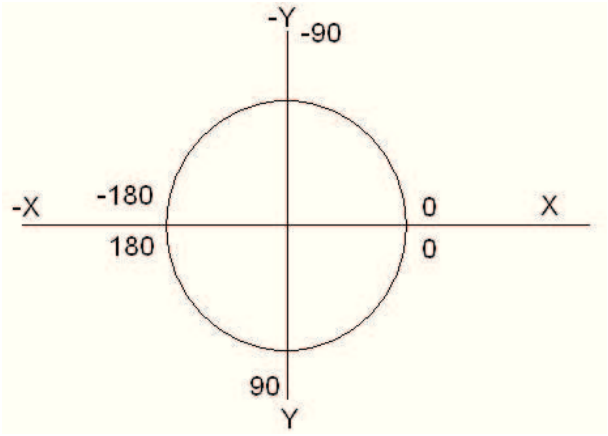
Chceme-li vytvořit taktiky pro roboty, musíme je sestavit na základě rozhodovacích bodů. Ty vychází z myšlenky popisu situace na hřišti, která je založena na několika vlastnostech. Pro každou vlastnost probíhá vždy unikátní výpočet nad reálnými daty, které získáme z analýzy obrazu. Cílem vlastnosti je poskytnout informaci, zda robot je ve stavu její realizace nebo ohrožuje-li míč naší bránu. Na výstupu dostaneme pravdivostní hodnotu se dvěma možnými stavy: $true$ je-li vlastnost realizovatelná a naopak $false$ není-li realizovatelná. Tvorba vlastností má své opodstatnění. Je to podstata tvorby taktik. Kombinací těchto vlastností získáme rozhodovací body, podle kterých můžeme robotům přidělovat akce. Pro výpočty je nutno stanovit hrací stranu našeho týmu. Mnoho výpočtu vychází ze souřadnice branek a úhlů, které jsou pro každou stranu jedinečné. Naše hřiště je rozděleno na dvě poloviny vzhledem k ose x, kde hodnota 0 reprezentuje střed hřiště. Osa y nabývá kladných respektive záporných hodnot pod osou x respektive nad osou x. Změnou souřadnic y osy dochází k přehození souřadnicového systému a je potřeba s touto změnou počítat u všech výpočtu. Změna sebou nese taky změnu v hodnotách úhlů. Ty jsou rozděleny na dvě poloviny a to $< 0; 180 >$, $< 0; -180 >$, které nabývají kladných respektive záporných hodnot pod osou x respektive nad osou x. Analýza obrazu získané úhly natočení robota počítala v normálním souřadnicovém systému. To vedlo k chybám výpočtu. Jakmile byla chyba odhalena, její odstranění řešilo od 360° odečíst získaný úhel. Tímto jsme převedli úhel do našeho souřadnicového systému. Náš tým bude na hrací ploše vždy považován, za hrající na levé straně. Na obrázku 10 je vyobrazen souřadnicový systém. Globální vlastnosti reprezentují veškeré dění na hřišti, které je společné pro obě strany a všechny roboty. Můžeme říci, že souvisí se všemi objekty, které se nachází aktuálně na hřišti. Vlastnosti robota se zabývají přímo konkrétním robotem a určují jeho možnosti a schopnosti, které je schopen v aktuálním okamžiku vykonat. Informace k vlastnostem byly čerpány z [4].

Seznam vlastností, které jsou podstatou volby akce robota.
Globální vlastnosti:

- tlak míče - poukazuje na možné ohrožení ze strany míče,
- míč na soupeřově straně - zda poloha míče je na straně soupeře,
- míč na naši straně - zda poloha míče je na naší straně,
- pohyb do naší branky - zda míč se pohybuje do naší brány,
- pohyb na naší polovinu - zda míč směřuje k naší polovině,
- míč u naší branky - zda se míč nachází blízko naší braný.

Vlastnosti robota:

- má míč - zda robot kontroluje pohyb míče,

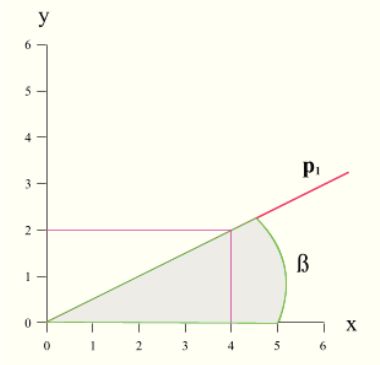


Obrázek 10: Souřadnicový systém

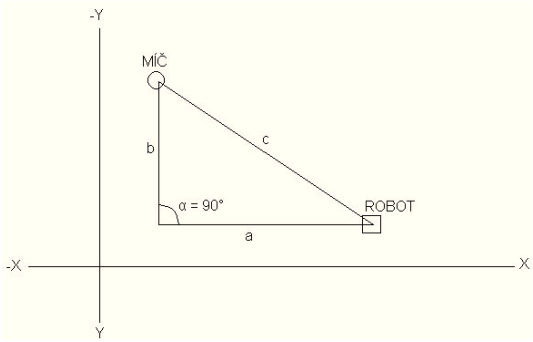
- může vystřelit - zda robot je ve střelecké pozici,
- volný prostor na bránu- zda má volný prostor směrem k bráně soupeře,
- pravděpodobnost vystřelení - zda se míč pohybuje v takovém směru, že robot bude moct střílet,
- mantinel - zda robot má příležitost, střílet na branku soupeře odrazem od horního nebo dolního mantinelu,
- střela z otočky - zda se robot nachází v blízkosti branky soupeře, tak aby mohl z otočení vystřelit,
- otáčení - zda se má robot otáčet na místě,
- trojúhelník branky - detekce predikované šance skórovat,
- odkop - zda je možno odkopnout míč z naší poloviny během bránění,
- pohyb brankáře - určuje kam se má brankář pohybovat.

5.1 Popis vlastností

Než se dostaneme k samotným vlastnostem, je potřeba vysvětlit důležitou funkci, s kterou počítáme u všech výpočtů. Veškeré počítání se neobejde bez zjištění úhlu, ať už mezi objekty nebo jednotlivým robotem a brankovištěm. K tomuto účelu byla použita C++ knihovna *math*, která nabízí funkci $atan2$. Funkce $atan2$ počítá arkustangens dvou proměnných x a y. Výpočet je podobný výpočtu arkustangens $\frac{y}{x}$, kromě toho, že znaménka obou argumentů jsou použita pro určení kvadrantu, ve kterém se nachází výsledná hodnota. Jinak řečeno, vrátí arkustangens zadaných souřadnic, to znamená úhel, mezi vodorovnou osou x a přímkou procházející body $(0, 0)$ a (x, y). Výsledek funkce je v radiánech,



Obrázek 11: Funkce atan2 [7]



Obrázek 12: Aplikovaná Pythagorova věta

který nabývá hodnot mezi $-\pi$ a π včetně, proto na závěr radiány převádíme na stupně, s kterými se dále pracuje. Funkce je vyobrazena na obrázku 11. Informace o této funkci byly čerpány z [6].

Druhou nezbytnou informací pro výpočty je vzdálenost mezi jednotlivými objekty. Tu jsme schopni určit pomocí Pythagorovy věty. Pythagorova věta popisuje vztah, který platí mezi délkami stran pravoúhlých trojúhelníků. Umožňuje dopočítat délku třetí strany takového trojúhelníku, pokud jsou známy délky dvou zbývajících stran. Vzdálenost například mezi robotem a míčem představuje trojúhelník, kde vzdálenost na ose x reprezentuje stranu a, vzdálenost na ose y reprezentuje stranu b a výsledná vzdálenost reprezentuje přeponu c. Úhel mezi stranou a a b je vždy 90°. Názornou ukázku reprezentuje obrázek 12.

V následujícím textu a ve výpočtech se vyskytuje spoustu důležitých konstant, kterými můžeme nastavit reakce a chování robota. Pro lepší přehlednost uvedu seznam všech použitých konstant s vysvětlením.
Konstanty pro vlastnosti robota:

- $maxControledAngle$ - použití ve vlastnostech: Má míč. Určuje úhel pod jakým má robot míč pod kontrolou.

- $maxAngle$ - použití ve vlastnostech: Může vystřelit, Odkop. Určuje úhel maximální odchylky míče od směru střely robota, ve kterém je schopen vystřelit.
- $maxRadius$ - použití ve vlastnostech: Může vystřelit. Určuje vzdálenost robota od míče, ve které je schopen vystřelit.
- $maxGoalieRadius$ - použití ve vlastnostech: Volný prostor na bránu, Mantinel. Určuje brankářskou oblast kolem brány.
- $possessionRange$ - použití ve vlastnostech: Má míč. Určuje vzdálenost, ve které robot má míč pod kontrolou.
- $radius$ - použití ve vlastnostech: Pravděpodobnost vystřelení, Mantinel. Určuje poloměr kolem bodu P.
- $runUp$ - použití ve vlastnostech: Pravděpodobnost vystřelení, Mantinel. Určuje vzdálenost bodu P od robota.
- $maximumHitDistance$ - použití ve vlastnostech: Otáčení, Střela z otočky. Určuje oblast, kdy je robot schopen otáčením zasáhnout míč.
- $maxInRange$ - použití ve vlastnostech: Odkop. Určuje maximální vzdálenost robota od míče pro obranný zákrok.
- $pathWidth$ - použití ve vlastnostech: Volný prostor na bránu. Určuje minimální vzdálenost překážky od cesty na bránu.
- $maxGoalieAngle$ - použití ve vlastnostech: Může vystřelit, Pravděpodobnost vystřelení. Určuje maximální odchylku natočení robota od centra soupeřovy brány.
- $maxBiliardRobotAngle$ - použití ve vlastnostech: Mantinel. Určuje maximální odchylku natočení robota od místa, kde by mněl střílet na mantinel.
- $maxPredictedShotAngle$ - použití ve vlastnostech: Trojúhelník branky. Určuje odchylku natočení od předvídaného míče.
- $maxBiliardAlfaAngle$ - použití ve vlastnostech: Mantinel. Určuje maximální rozdíl mezi alfou a úhlem, pod kterým se nachází robot vzhledem k místu, na které má střílet.
- $veryClose$ - použití ve vlastnostech: Střela z otočky. Určuje vzdálenost robota od branky soupeře, kdy je považován za velmi blízko.
- $maxShotSpinAngle$ - použití ve vlastnostech: Střela z otočky. Určuje úhel, pod kterým jde ještě mimo tyče branky vystřelit z otočky.
- $farDistance$ - použití ve vlastnostech: Pohyb brankáře. Určuje pro brankáře vzdálenost, od kdy míč ohrožuje bránu.

Konstanta	Hodnota
$maxControledAngle$	40
$maxAngle$	10
$maxRadius$	25
$maxGoalieRadius$	10
$possessionRange$	7
$radius$	10
$runUp$	25
$maximumHitDistance$	7
$maxInRange$	25
$pathWidth$	15
$maxGoalieAngle$	15
$maxBiliardRobotAngle$	15
$maxPredictedShotAngle$	20
$maxBiliardAlfaAngle$	5
$veryClose$	20
$maxShotSpinAngle$	45
$farDistance$	50
$closeDistance$	30

Tabulka 8: Hodnoty konstant

Konstanty pro globální vlastnosti:

- $closeDistance$ - použití ve vlastnostech: Míč u naší branky. Určuje od jaké vzdálenosti je míč považován za velmi blízko naší brance.

Hodnoty konstant byly odvozeny z testů jednotlivých vlastností. Postup testování je uveden v souhrnu vlastností. Tabulka 8 zobrazuje hodnoty konstant. Zaměříme-li se na hodnoty, neunikne nám, že v případě vzdáleností se jedná o centimetry. Na tom samozřejmě není nic špatného. Takto definované vzdálenosti nám jsou dobře známy. Je potřeba počítat, že analýza obrazu [3] všechny informace ukládá v pixelech. Takže během výpočtů, musíme brát v úvahu, že se jedná o vzdálenosti a souřadnice v těchto jednotkách. Budeme-li se pokoušet porovnat získanou vzdálenost s některou z výše definovaných konstant, obdržíme samozřejmě špatný výsledek. Abychom těmto situacím předešli, bylo potřeba vymyslet, jak převádět pixely na centimetry a obráceně. Vznikly tyto dvě metody 8, 9.

```
double RobotFeatures::XY_to_Cm(double xy)
{
   return (xy/WidthOfRobotPx)*WidthOfRobotCm;
}
```

Výpis 8: Metoda pro převod px na cm

```
double RobotFeatures::XY_to_Px(double xy)
{
    return (xy/WidthOfRobotCm)*WidthOfRobotPx;
}
```

Výpis 9: Metoda pro převod cm na px

Metody počítají se dvěma konstantami, kterými jsou $WidthOfRobotCm$, $WidthOfRobotPx$. První konstanta reprezentuje reálnou šířku robota v centimetrech a druhá reálnou šířku v pixelech. První konstantu je potřeba upravit pouze rozhodne-li se náš tým pro změnu robotu, kteří můžou mít trochu odlišné parametry. Konstantu $WidthOfRobotPx$ je potřeba kalibrovat při každé hře, kdy jsme pohybovali s umístěním kamery. Podle výšky umístění kamery nad hrací plochou bude docházet, ke změně šířky robota v pixelech. Jakmile máme nastaveny obě konstanty, jednotlivé metody jsou schopny převádět hodnoty. Metoda pro převod pixelů na vstup obdrží hodnotu v pixelech. Hodnota je vydělena šířkou robota v pixelech. Takto získáme kolikrát se délka robota v tomto čísle nachází. Zbývá jen vydělené číslo vynásobit šířkou robota v centimetrech. Získáme převedenou hodnotu z pixelů na centimetry. V případě převodu centimetrů na pixely dochází jen k přehození konstant.

5.2 Globální vlastnosti

Výpočet těchto vlastností realizuje výpis 13.

5.2.1 Tlak míče

Tlak míče v každém momentu určuje velikost ohrožení naší brány ze strany míče. Ohrožení se bude zvětšovat na základě tří parametrů: se zmenšující se vzdáleností míče od naší brány, na základě směru míče k naší bráně a v závislosti na poloze na ose x. Největší prioritu má poloha na ose x, protože z ní vyplývá největší ohrožení. Bude-li se míč nacházet blízko naší brány, je to již dostatečný podnět k ohrožení. Natočení a vzdálenost míče slouží jako modifikátory pro vzdálenosti, kdy míč není v neprostření blízkosti brány. Směřuje-li míč do brány, roboti o tom budou dostatečně rychle varování, aby mohli reagovat a ohrožení eliminovat. Ohrožení na základě vzdálenosti od brány začíná nabývat hodnot, jakmile míč překročí středovou čáru a pohybuje se na naší polovině hřiště. Dosáhne-li ohrožení 60 a více procent, výstupem bude hodnota $true$. V opačném případě ohrožení není kritické a na výstupu dostaneme $false$.
Rozdělení podle priority:

- 20% závisí na směru míče. Směr od naší brány znamená menší ohroženi.
- 20% závisí na blízkosti míče k naší brance.
- 60% závisí na souřadnici na ose x.

5.2.2 Míč na soupeřově straně

Je-li souřadnice míče na ose x větší než je nula, pak míč se nachází na soupeřově straně hřiště. Vlastnost bude označena za $true$. V opačném případě $false$.

5.2.3 Míč na naší straně

Je-li souřadnice míče na ose x menší než je nula, pak míč se nachází na naší straně hřiště. Vlastnost bude označena za $true$. V opačném případě $false$.

5.2.4 Pohyb do naší branky

Tato funkce detekuje, zda se míč pohybuje takovým způsobem, že by mohl ohrozit naší branku. Pokud předpokládaná trajektorie míče překročí brankovou čáru, pak je vlastnost označena za $true$, v opačném případe $false$.

5.2.4.1 Algoritmus: Od souřadnic horní tyče branky odečteme souřadnice míče. Pomocí funkce $atan2$ získáme úhel pod jakým se nachází míč k tyči. Stejný výpočet provedeme pro dolní tyč. Na závěr porovnáme rotaci míče se získanými úhly. Nachází-li se rotace mezi úhlem horní a dolní tyče, pak míč se pohybuje pod takovým úhlem, že by překročil brankovou čáru.

5.2.5 Pohyb na naší polovinu

Tato funkce určuje zda se míč pohybuje k naší nebo soupeřově bráně. Výstupem vlastnosti je $true$, pohybuje-li se míč směrem k naší brance, $false$ pohybuje-li se směrem k brance soupeře.

5.2.5.1 Algoritmus: Je-li orientace míče $< -90°$ nebo $> 90°$ hodnota je $true$. Je-li orientace míče $> -90°$ nebo $< 90°$ hodnota je $false$.

5.2.6 Míč u naší branky

Jakmile se vzdálenost míče od středu naší branky stane menší než konstanta $closeDistance$, funkce vrací hodnotu $true$, jinak $false$.

5.2.6.1 Algoritmus: Od souřadnic středu naší branky odečteme souřadnice míče. Takto získáme vzdálenosti pro osu x a y. Pomocí Pythagorovy věty spočítáme přeponu, která představuje vzdálenost míče od středu brány. Vzdálenost je v pixelech, proto ji musíme pomoci metody XY_to_Cm převést na centimetry a teprve nyní porovnat s konstantou $closeDistance$.

5.3 Popis vlastností robota

Nyní se zaměříme na vlastnosti robota, které jsou základním prvkem tvorby taktik.

5.3.1 Má míč

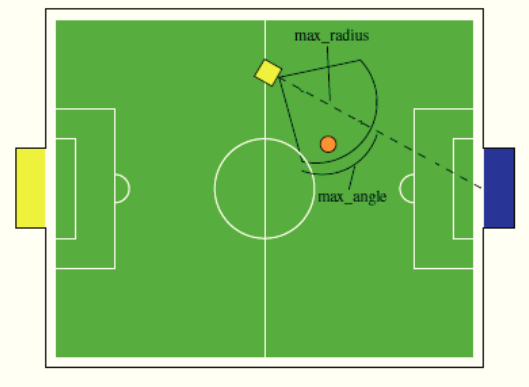
Cíl této vlastnosti je velmi prostý. Má určit, kdy robot kontroluje pohyb míče. Toto je však velmi neurčité a těžko definovatelné. Na problematiku můžeme nahlížet, jako na vzdálenost mezi míčem a robotem. Z prováděných testů vyplynulo, že založení vlastnosti na vzdálenosti je nedostačující. Míč je považován pod kontrolou, když se nachází z boku, či zezadu robota. To samozřejmě s kontrolou nemá nic společného jelikož, robot je nucen otočit se a během tohoto časového intervalu nebude s míčem v kontaktu. Z tohoto důvodu přibyla detekce natočení robota vzhledem k míči. Pokud se nachází míč ve vzdálenosti do konstanty $possessionRange$ a zároveň odchylka úhlu nepřesáhne konstantu $maxControledAngle$, pak výsledkem je hodnota $true$, jinak $false$.

5.3.1.1 Algoritmus (výpis 14): Od souřadnic míče odečteme souřadnice robota. Takto získáme vzdálenosti mezi objekty na jednotlivých osách. Z Pythagorovy věty spočítáme vzdálenost robota od míče. Od natočení robota odečteme natočení míče, to získáme pomocí metody $atan2$. Vzdálenost pomocí metody XY_to_Cm převedeme na centimetry. Na závěr provedeme porovnání vzdálenosti a rozdílného úhlu s konstantami. V případě splnění podmínek vlastnost vrací $true$.

5.3.2 Může vystřelit

Cílem této vlastnosti je detekce zda robot má příležitost vystřelit na bránu soupeře. Tato vlastnost jako jediná, při detekci střely počítá s reálnými souřadnicemi míče. Další budou vycházet z predikovaných souřadnic. Princip je založen na předpokladu, že se jedná o kruhovou výseč, v jejímž středu se nachází robot. Velikost výseče je dána úhlem 2 * konstanta $maxAngle$. Maximální vzdálenost míče od robota reprezentuje poloměr výseče, tj. konstanta $maxRadius$. Střed výseče je vždy natočen směrem na centrum soupeřovy brány. Nachází-li se míč uvnitř této výseče, pak vlastnost vrací hodnotu $true$, v případě polohy mimo výseč vrací $false$. V průběhu testů byl zaznamenán nedostatek této myšlenky. Musíme vzít v úvahu taky směr natočení robota. I když jsou splněny předešlé podmínky, výsledek je fatální, když robot je nasměrován zcela jiným směrem než je branka. Po detekci této vlastnosti, že může střílet, lze očekávat zrychlení robota. Bude-li natočeny opačně, tak pouze se vzdálí od míče. Algoritmus byl doplněn o detekci, zda robot je natočen ve směru branky, na kterou bude střílet. Vlastnost zobrazuje obrázek 13.

5.3.2.1 Algoritmus (výpis 15): Od souřadnic centra soupeřovy brány odečteme souřadnice robota. Ze získaných délek pomocí metody $atan2$ spočítáme natočení brány. Od tohoto úhlu odečteme rotaci robota a výsledek představuje odchýlení natočení robota od brány. Ten porovnáme s konstantou $maxGoalieAngle$. Je-li splněna tato podmínka, pak robot směřuje na bránu. Nyní musíme zjistit zda míč se nachází ve výseči. Od souřadnic míče odečteme souřadnice robota. Ze vzniklých délek spočítáme metodou $atan2$ polohu míče vůči robotovi a Pythagorovou větou vzdálenost míče od robota.Vzdálenost pomocí metody XY_to_Cm převedeme na centimetry. Zjistíme zda míč se nenachází dál, než je konstanta $maxRadius$. V předchozích krocích jsme získali dva úhly: polohu brány a míče



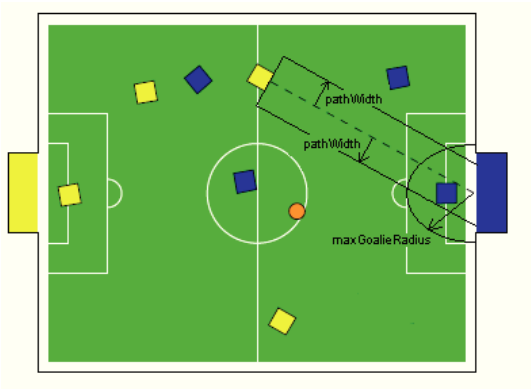
Obrázek 13: Ukázka vlastnosti Může vystřelits [4]

vůči robotovi. Odečtení úhlu míče od úhlu brány, představuje vzdálenost míče od středu výseče. Nepřesáhne-li velikost konstanty $maxAngle$, pak robot se nachází uprostřed výseče. Jsou-li splněny všechny podmínky, pak víme, že robot je nasměrován na bránu a míč se nachází uprostřed výseče. Robot je schopen vystřelit.

5.3.3 Volný prostor na bránu

Cílem vlastnosti je detekce, zda robot je schopen vystřelit nebo přesunout se k bráně, aniž by mu v cestě stála nějaká překážka. Vlastnost nesmí brankáře soupeře považovat za překážku, protože by jinak nikdy nenabyla hodnoty $true$. Brankář se v bráně pohybuje po celou dobu zápasu. Užitečnost této vlastnosti spočívá ve zjištění, zda robot je nějakým způsobem bráněn soupeřovými roboty. Zvažuje překážky na cestě o šířce $pathWidth$. Za cestu je považována vzdálenost mezi robotem a centrem soupeřovy brány. Všichni roboti, kteří se budou nacházet blíže brankové čáře, než je konstanta $maxGoalieRadius$, budou vyhodnoceni jako brankáři a do výpočtu je nezahrneme. Výpočet provedeme nad všemi objekty . Nachází-li se alespoň jedna překážka uvnitř této cesty, vlastnost vrátí $false$. Naopak jsou-li všechny překážka mimo cestu, vlastnost vrátí $true$. Je potřeba uvědomit si, že v tomto případě překážku pro robota nepředstavuje pouze robot soupeře, ale také jeho spoluhráči. V průběhu testování došlo k úpravě metody a to tak, aby mohla detekovat, zda robot má volný prostor na libovolný bod na hřišti. Vlastnost musí být nastavena, zda se jedná o prostor na bránu nebo na pozici. V případě brány nepočítá s brankářem. Vlastnost zobrazuje obrázek 14.

5.3.3.1 Algoritmus (výpis 16): Následující krok má význam pouze za předpokladu, že se jedná o pohyb na bránu a je potřeba ignorovat brankáře. Od souřadnic překážky odečteme souřadnice centra soupeřovy brány. Z takto spočítaných vzdáleností mezi objekty, pomocí Pythagorovy věty zjistíme vzdálenost překážky od brány. Vzdálenost pomocí metody XY_to_Cm převedeme na centimetry. Je-li tato vzdálenost menší, než konstanta $maxGoalieRadius$, pak se jedná o brankáře a tento robot je ignorován. Nyní zjišťujeme,



Obrázek 14: Ukázka vlastnosti Volný prostor na bránu [4]

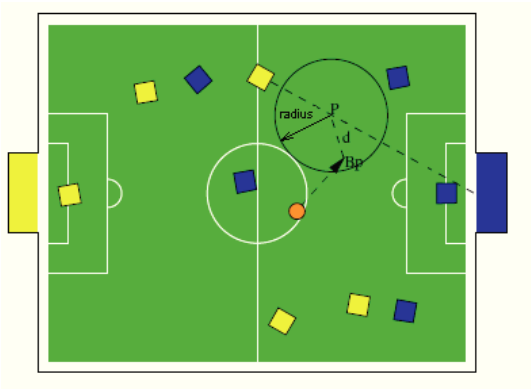
zda se překážka nachází mezi robotem a cílem. Souřadnice překážky na ose x se vždy musí nacházet mezi souřadnicemi robota a cíle. Nenachází-li se, pak není schopna ohrozit robota na cestě k cíli. Nyní je potřeba pro robota určit úhel mezi překážkou a bránou. Od souřadnic robota odečteme souřadnice brány. Ze získaných vzdálenosti na jednotlivých osách, pomocí metody $atan2$, spočítáme úhel mezi robotem a bránou. Pak od souřadnic robota odečteme souřadnice překážky. Opět pomocí metody $atan2$ spočítáme úhel mezi robotem a překážkou. Pythagorovou větou získáme vzdálenost robota od překážky. Abychom dostali úhel mezi překážkou a bránou, musíme předchozí úhly od sebe odečíst. Výsledný úhel pro přehlednost nazvěme α a převedeme jej na radiány. Souřadnice robota, překážky a cesty reprezentují pravoúhlý trojúhelník, protože překážka se nachází vzhledem k cestě vždy pod úhlem 90°. Když známe úhel a délku jsme schopni vypočítat výšku, která reprezentuje vzdálenost překážky od cesty. Tu získáme ze vzorce:

$$vc = b * \sin \alpha$$

Na závěr vzdálenost převedeme pomocí metody XY_to_Cm na centimetry a porovnáme s konstantou $pathWidth$. Je-li menší než konstanta, vlastnost vrací $false$, protože v cestě robotovi se nachází překážka a nemá volný prostor k cíli.

5.3.4 Pravděpodobnost vystřelení

Cílem vlastnosti je zjistit, zda některý z naších nebo soupeřových robotů se v příštím kroku dostanou k příležitosti vystřelit na bránu. Na obrázku značka Bp reprezentuje predikovaný míč, který se na této pozici bude v následující době nacházet. Bod P se nachází na čáře mezi robotem a centrem brány. Jeho pozice je ve vzdálenosti konstanty $runUp$ od robota. Poloměr kružnice kolem bodu P má velikost rovnou konstantě $radius$. Nechť d je vzdálenost mezi bodem P a predikovanou souřadnicí míče Bp. Pak vlastnost vrací hodnotu $true$, za předpokladu, že d je menší, než hodnota konstanty $radius$, jinak vrací $false$. Tak jako u vlastnosti detekující zda může vystřelit, i zde je potřeba počítat s natočením robota k bráně. Je-li otočen jiným směrem, není schopen střely. Vlastnost zobrazuje obrázek 15.



Obrázek 15: Ukázka vlastnosti Pravděpodobnost vystřelení [4]

5.3.4.1 Algoritmus (výpis 17): Voláme-li tuto metodu, musíme pomocí parametru our definovat, zda se jedná o robota našeho nebo soupeře. Každý bude vyhodnocován k jiným souřadnicím brány. Roboti soupeře pracují se souřadnicemi naší branky a naopak naší roboti se souřadnicemi brány soupeře. V prvním kroku určíme jestli je robot natočený k bráně. Od souřadnic brány odečteme souřadnice robota. Ze vzdáleností na jednotlivých osách, pomocí metody $atan2$, spočítáme úhel mezi robotem a bránou. Abychom mohli ověřit natočení robota, musíme získaný úhel odečíst od jeho rotace. Tento úhel nazvěme α. Je-li α menší, než je konstanta $maxGoalieAngle$, pak robot směřuje k bráně. V druhém kroku potřebujeme určit polohu bodu P. Konstanta $runUp$ mezi robotem a tímto bodem představuje přeponu pravoúhlého trojúhelníku. Pomocí metody XY_to_Px ji převedeme z centimetrů na pixely. Ze vzorců

$$a = \sin\alpha * c$$
$$b = \cos\alpha * c$$

spočítáme, jak daleko se nachází bod P od robota na jednotlivých osách. Samotné délky nám neposkytnou žádnou informaci. Musíme je připočítat k souřadnicím robota. Takto získáme souřadnici bodu P na hřišti. Posledním krokem je výpočet vzdálenosti predikovaného míče od tohoto bodu. Od souřadnic bodu P odečteme souřadnice míče. Ze získaných vzdálenosti, Pythagorovou větou spočítáme vzdálenost, kterou převedeme pomocí metody XY_to_Cm na centimetry. Provedeme porovnání, jestli vzdálenost je menší, než konstanta $radius$. Je-li tomu tak, pak vlastnost vrátí hodnotu $true$.

5.3.5 Mantinel

Cílem vlastnosti je detekce, zda robot má příležitost střílet na bránu soupeře odrazem od horního nebo dolního mantinelu. Detekce je založena na podobném výpočtu jako vlastnost Pravděpodobnost vystřelení. V tomto případě se však střed kruhu nachází ve zcela jiném směru, protože cílem jíž není přímá střela na branku, ale pokusit se střelit gól prostřednictvím odrazu od jednoho z mantinelů. Míč dopadající pod úhlem na mantinel

se pod tímto úhlem od mantinelu také odrazí. Je potřeba určit pozici, kam má robot míč vystřelit, tak aby po odrazu míč směřoval do brány soupeře. Provedení takového výstřelu je velmi složité a náročné na rychlost robota. Aby střela měla alespoň nějakou šanci ohrozit soupeřovu bránu, je zapotřebí vyvinout dostatečnou rychlost. Vlastnost zobrazuje obrázek 16.

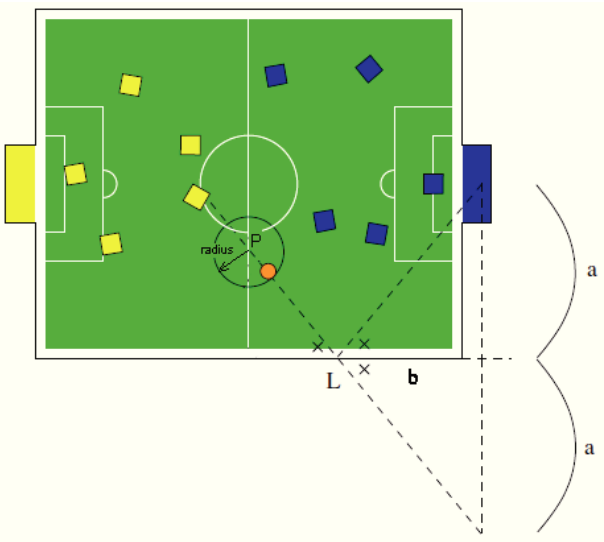
5.3.5.1 Algoritmus (výpis 18): Nejdříve zjišťujeme, zda se robot nachází na polovině soupeře a jestli není příliš blízko jeho brány. Robot nesmí být blíže brance, než je hodnota konstanty $maxGoalieRadius$. Potřebujeme určit úhel, pod kterým se míč musí odrazit od mantinelu, aby směřoval do středu soupeřovy brány. K jeho určení potřebujeme znát délku a a b. Od souřadnice x robota odečteme souřadnici x branky. Výslednou hodnotu vydělíme dvěma. Tím obdržíme délku strany b. Podle polohy míče počítáme s horní nebo dolní y souřadnicí mantinelu. Od souřadnice y mantinelu odečteme souřadnici y branky. Tím obdržíme délku strany a. Pomocí metody $atan2$ vypočítáme úhel α, který představuje potřebný úhel dopadu a odrazu míče. Nyní potřebujeme znát natočení robota k místu odrazu. Délku na ose x představuje také strana b. Délku pro osu y spočítáme odečtením souřadnice y robota od souřadnice y mantinelu. Metodou $atan2$ získáme úhel $tempAngle$, který určuje pod jakým úhlem se nachází robot vzhledem k místu odrazu. Odečteme-li od $tempAngle$ rotaci robota, získáme rozdílný úhel natočení. Nepřesáhne-li tento rozdíl konstantu $maxBiliardRobotAngle$, víme že robot je natočen tím směrem. Pouhé natočení robota není dostačující, ten se musí nacházet vzhledem k místu odrazu pod úhlem α. Jinak by nebyl schopen vystřelit míč požadovaným směrem. Odečteme-li od úhlu α úhel $tempAngle$, dostaneme rozdíl oproti požadovanému úhlu dopadu. Nepřesáhne-li rozdíl konstantu $maxBiliardAlfaAngle$, pak robot se nachází pod úhlem, pod kterým střela bude směřovat do brány. Nyní víme, že robot je natočený směrem k místu odrazu a také se nachází pod správným úhlem. Je potřeba určit souřadnici bodu P. Konstantu $runUp$ pomocí metody XY_to_Px převedeme na pixely. Tato konstanta představuje přeponu, strana a představuje délku na ose y a strana b délku na ose x. Ze vzorců

$$b = \cos\alpha * c$$
$$a = \sin\alpha * c$$

spočítáme jednotlivé délky, které přičteme k souřadnicím robota. Výsledkem je souřadnice bodu P. Zbývá určit vzdálenost míče od tohoto bodu. Od souřadnic bodu P odečteme souřadnice míče. Pythagorovou větou spočítáme tuto vzdálenost, kterou pomocí metody XY_to_Cm převedeme na centimetry. Výslednou hodnotu porovnáme s konstantou $radius$. Je-li menší, pak míč se nachází uprostřed kruhu. Jsou-li splněny všechny podmínky robot je schopen vystřelit na bránu odrazem od mantinelu.

5.3.6 Otáčení

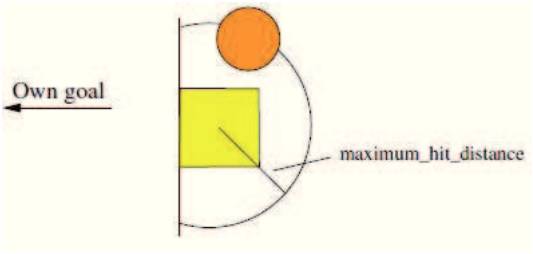
Cílem funkce je detekce, kdy se má robot začít otáčet na místě. K otáčení by mělo dojít, kdykoliv je míč v dosahu, aby jej robot zasáhl. Abychom předešli situacím, kdy se robot začne otáčet později, výpočet vychází z predikovaného míče. Víme, že míč v příštím



Obrázek 16: Ukázka vlastnosti Mantinel [4]

kroku bude v dosahu zásahu a robot se začne dostatečně brzy otáčet. Tím je dosaženo dostatečné rychlosti k odrazu a také nenastane situace, že by propásl svoji šanci. Odrazem musí míč směřovat opačným směrem, než je poloha naší brány. Nachází-li se střed míče ve vzdálenosti konstanty $maximumHitDistance$ od robota, pak je schopen míč zasáhnout. Tato konstanta je dána součtem poloviny úhlopříčky robota a poloměru míče. Jedná se tedy o maximální vzdálenost, kdy je možné míč trefit. Tato situace je vyobrazena na obrázku 17. Výsledkem této situace je $true$, míč je možné odrazit pryč. K otáčení nesmí dojít, nachází-li se míč za robotem směrem k naší brance. Odražení u této situace, povede k odrazu směrem k naší bráně nebo v horším případě zapříčiní střelení vlastního gólu. Vlastnost je především určena, pro našeho brankáře, který se takto stává velmi efektivním. Snažíme-li se dostat míč z našeho brankoviště, dochází k odkopu míče, aniž by brankář musel opustit branku. Opuštění brány, je vždy velmi riskantním krokem.

5.3.6.1 Algoritmus (výpis 28): Nejdříve se musíme přesvědčit, jestli se míč nenachází za robotem. Souřadnici míče na ose x porovnáme s x-ovými souřadnicemi robota a centra naší brány. Nachází-li se souřadnice míče mezi nimi, pak je míč za robotem a není možné provést vyražení otáčením. Není-li tomu tak, spočítáme vzdálenosti mezi objekty pro jednotlivé osy. Od souřadnic robota odečteme souřadnice míče. Ze získaných délek pomocí Pythagorovy věty dostaneme vzdálenost míče od robota. Tu pomocí metody XY_to_Cm převedeme na centimetry a porovnáme s konstantou $maximumHitDistance$. Je-li vzdálenost menší, než tato konstanta, pak míč se nachází v prostoru, kdy je možné jej zasáhnout. Vlastnost vrací $true$.



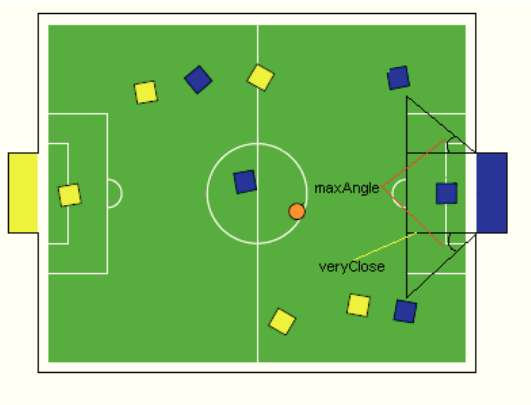
Obrázek 17: Ukázka vlastnosti Otáčení [4]

5.3.7 Střela z otočky

Tato vlastnost částečně vychází z předešlé. Zatímco Otáčení je spíš určena pro brankáře, Střela z otočky je navržena útočníkovi. I zde musíme vycházet z predikovaných souřadnic míče, aby k otáčeni robota došlo dostatečně brzy a mohl získat dostačující rychlost pro střelu. Aby vlastnost nabyla hodnoty $true$ musí být splněno několik podmínek. Především efektivnost této střely se s zvětšující vzdálenosti od brány snižuje. Robot se musí tedy nacházet maximálně ve vzdálenosti konstanty $veryClose$. Z obrázku 18 je vidět plochu reprezentující střelecké pozice. Nachází-li se robot mimo brankoviště, jinými slovy jeho souřadnice y se nenachází mezi tyčemi brány. Zjišťujeme úhel, pod jakým se k nejbližší tyčí nachází. Jestliže velikost úhlu nepřesáhne konstantu $maxAngle$, pak robot má pořád šanci na úspěšnou střelu. V opačném případě je vracena hodnota $false$. S dobrou pozicí a míčem v dosahu, bychom nevystačili. Musíme předpokládat, že míč může být kdekoliv kolem robota. Poloha míče se musí nacházet před robotem, aby během otáčení směr střely směřoval do branky a nedošlo k nechtěnému odkopu směrem na naší stranu. O to je to podstatnější, nachází-li se robot mimo brankoviště. Zde je ještě podmínkou, aby míč byl pod robotem respektive nad robotem, podle pozice, u které tyče se nachází horní respektive dolní.

5.3.7.1 Algoritmus (výpis 20): V prvním kroku musíme určit, zda je robot blízko brány, zda je míč před robotem a je-li možné jej zasáhnout. Nejsou-li splněny všechny podmínky, robot není schopen vystřelit otáčením. Od x souřadnice brány soupeře odečteme x souřadnici robota. Výslednou délku pomocí metody XY_to_Cm převedeme na centimetry a porovnáme s konstantou $veryClose$. Je-li vzdálenost menší, pak robot se nachází blízko brány. Když je x souřadnice míče větší, než x souřadnice robota, pak míč se nachází před ním. Zbývá určit, zda je míč v dosahu. Od souřadnic míče odečteme souřadnice robota. Ze vzniklých délek pomocí Pythagorovy věty spočítáme vzdálenost míče od robota. Tu pomocí metody XY_to_Cm převedeme na centimetry a porovnáme s konstantou $maximumHitDistance$. Je-li vzdálenost menší, pak je možné míč trefit. Mohou nastat tyto tři situace:

1. Pozice robota je mezi tyčemi brány.
2. Robot je mimo horní tyč.



Obrázek 18: Ukázka vlastnosti Střela z otočky [4]

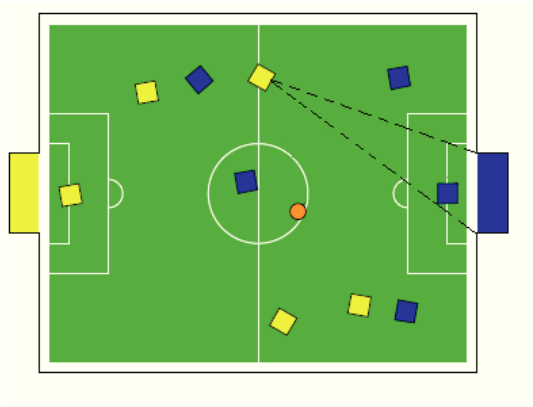
3. Robot je mimo dolní tyč.

Nastane-li první případ vlastnost vrací $true$. Pro zbývající dvě je ještě potřeba určit, zda se nenachází ve střeleckém úhlu, kdy má pořád šanci na střelu. Zároveň míč se musí nacházet vždy nad respektive pod robotem v závislosti, u které tyče se nachází. Tím zabráníme vystřelení míče do rohu hřiště. Podle pozice robota budeme počítat s jednou z brankových tyčí. Od souřadnic této tyče odečteme souřadnice robota. Pomocí funkce $atan2$ získáme úhel natočení brány vzhledem k robotovi. Úhel porovnáme s konstantou $maxShotSpinAngle$ a zjistíme, jestli míč je směrem k bráně. Za předpokladu splnění obou podmínek, pak vlastnost pro situaci 2 nebo 3 nabývá hodnoty $true$.

5.3.8 Trojúhelník

Vlastnost detekuje možnost střely na bránu. Jedná se o další útočnou vlastnost, která vychází z predikovaných souřadnic míče. Během hry se míč pohybuje na hřišti velice rychle a to může mít za následek zpoždění detekce, kdy má robot vystřelit. Cílem je redukovat počet takto promrhaných šancí. Jelikož vycházíme z dat, která jsou odvozena z aktuální situace na hřišti, je snahou poskytnout robotovi impuls dostatečně včas. Ten pak je schopen dostatečně brzo reagovat a nabrat odpovídající rychlost. V příštím kroku se bude pohybovat dostatečně rychle, aby provedl výstřel na bránu soupeře. Vlastnost zjišťuje, zda se míč v příštím časovém úseku objeví na pozici mezi robotem a jednotlivými okraji hranice soupeřovy brány. Pokud ano, pak vlastnost vrací hodnotu $true$ a předpokládáme, že robot se dostane do střelecké pozice. Proces vystřelení na bránu je započat trochu dříve, tím dojde ke snížení počtu promrhaných střel. Vlastnost zobrazuje obrázek 19.

5.3.8.1 Algoritmus (výpis 21): Jako u každé střelecké vlastnosti i zde hraje důležitou roli natočení robota na svůj cíl. Cíl je reprezentován míčem. Od souřadnic míče odečteme souřadnice robota. Ze vzniklých délek pomocí funkce $atan2$ spočítáme úhel α, který představuje polohu míče vůči robotovi. Abychom určili natočení robota k míči, odečteme



Obrázek 19: Ukázka vlastnosti Trojúhelník [4]

od tohoto úhlu jeho rotaci. Nepřesáhne-li výsledek konstantu $maxPredictedShotAngle$, pak natočení robota je směrem k míči. Abychom určili, zda míč se nachází uprostřed trojúhelníku, potřebujeme znát natočení tyčí branky vůči robotovi. Pro každou tyč provedeme následující kroky. Od souřadnic tyče odečteme souřadnice robota. Ze vzniklých délek pomocí funkce $atan2$ získáme úhel natočení. Zbývá porovnat úhel natočení míče α s úhly tyčí. Nachází-li se α mezi těmito úhly, pak míč je uprostřed trojúhelníku a robot má příležitost k predikované střele.

5.3.9 Odkop

Cílem vlastnosti je detekce, kdy robot může vyrazit míč pryč od naší brány. Takto jsme schopni efektivně bránit prostor kolem brankoviště. Míč v tomto prostoru, představuje pro nás riziko a je lepší ho maximálně redukovat odkopem, než aby se robot snažil s míčem manévrovat před bránou. Vlastnost nabývá hodnoty $true$, jsou-li splněny následující tři podmínky:

1. Míč se nachází na naší polovině hřiště.
2. Míč je blíž robotovi, než je konstanta $maxInRange$.
3. Míč je na straně robota, natočeného směrem k brance soupeře.

5.3.9.1 Algoritmus (výpis 22): Chceme-li zjistit, zda jsou splněny výše zmíněné podmínky, potřebujeme znát vzdálenost míče od robota, jeho pozici vůči robotovi a také, je-li robot natočený směrem k míči. Od souřadnic míče odečteme souřadnice robota. Ze vzniklých délek pomocí Pythagorovy věty spočítáme vzdálenost mezi robotem a míčem. Výsledek převedeme pomocí metody XY_to_Cm na centimetry a porovnáme s konstantou $maxInRange$. Délky použijeme také pro výpočet úhlu natočení míče $angle$ vůči robotovi. K tomu použijeme funkci $atan2$. Pokud úhel $angle$ je větší, než -90° a zároveň menší, než 90°, pak míč je vůči robotovi ve směru na soupeřovu bránu. Zbývá

určit, zda robot je natočen k míči. Od úhlu $angle$ odečteme rotaci robota. Výsledná hodnota reprezentuje odchylku natočení. Jsou-li splněny všechny podmínky vlastnost vrací $true$.

5.3.10 Pohyb brankáře

Cílem vlastnosti je určit, kdy se má brankář pohybovat. Snahou je dosáhnout takové pozice, aby brankář pokryl čím jak nejvíce plochy před bránou a zároveň blokoval všechny možné střely soupeře. Běžná pozice brankáře je uprostřed brány. Nachází-li se míč v dostatečné vzdálenosti, pak není nutné s ním pohybovat. Jakmile se však míč dostane do vzdálenosti dané konstantou $farDistance$, robot se začne pohybovat po ose y.

5.3.10.1 Algoritmus (výpis 23): Zde nás zajímá pouze jediná informace a to, zda míč se nachází ve vzdálenosti konstanty $farDistance$. Od x souřadnice středu naší brány odečteme x souřadnici míče. Konstantu pomocí metody XY_to_Px převedeme na pixely a porovnáme se vzdáleností míče od brány. Nachází-li se míč dál, pak vlastnost vrací $false$, jinak vrací $true$.

5.4 Souhrn vlastností

Snahou bylo vytvořit sadu vlastností, s jejíž pomocí je uživatel schopen navrhnout rozhodovací body v taktikách. Vlastnosti zahrnují všechny možné situace, které během hry mohou na hřišti vzniknout a je potřeba je umět detekovat, aby robot byl schopen na ní reagovat. Jednotlivé vlastnosti byly testovány přímo na hřišti s pomocí kamery. O snímání dat se starala analýza obrazu [3]. Snahou bylo simulovat reálné prostředí, tak jako by se hrál opravdový zápas mezi roboty. Pro většinu testů jsme si vystačili pouze s jedním robotem. V případě detekce, zda má robot volný prostor byl použit ještě jeden, který reprezentoval překážku. Při testech jsme se z počátku velmi často potýkali s problémem, kdy na hřišti vznikaly „hluchá místa“, v kterých analýza obrazu nebyla schopna detekce objektů. Jejím kritickým bodem jsou prostory kolem mantinelů, v kterých jsme se potýkali téměř neustále s problémy. To má vliv na detekci vlastností v těchto prostorách. Testy byly prováděny s vypnutým robotem. Usnadňovalo nám to práci s daty, kdy v „debug“ módu jsme mohli krokovat neustále stejná data a hledat závady v algoritmech. Zároveň s robotem bylo možné pohodlně manipulovat a natáčet, za účelem vyzkoušení co možná nejvíce variant s úhly. Všechny vykonané testy vedly ke zdokonalení jednotlivých vlastností a detekce chyb v kódu. Tyto experimenty pomohly odvodit hodnoty konstant, pro jednotlivé vlastnosti a dosáhnout, tak přesných výsledků.

6 Dovednosti

Vše co bylo doposud v této práci uvedeno, souviselo spíše s tvorbou rozhodovacích bodů a taktikami samotnými. S tím si ovšem zdaleka nevystačíme. Robot sice ví jakou akci má provést, ale neví jak toho docílit. O korektní provedení zvolené taktiky se nám starají jednotlivé dovednosti. Bavíme-li se o dovednostech je potřeba vysvětlit, co to vlastně znamená. Jedná se tedy o nízko-úrovňové akce robota. Výstupem takové akce je nastavení pravého a levého kola, tak aby byl docílen požadovaný pohyb a směr. Robot na každou situaci bude trochu jinak reagovat. To znamená jiné nastavení rychlostí kol, proto nemůžeme mít pouze jedinou metodu, která by tento výpočet prováděla. Informace k dovednostem byly čerpány z článků [8], [1]. Chceme-li, aby naši roboti byli schopni porazit soupeře, musí zvládat následující dovednosti.
Seznam základních dovedností:

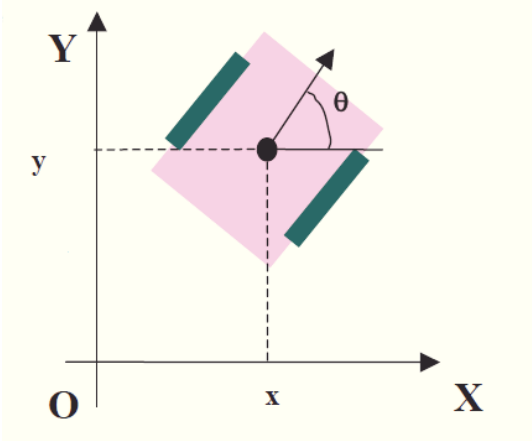
- jít za míčem - robot se snaží dostat k míči,
- tah na bránu - robot se snaží dostat s míčem k bráně soupeře,
- střela - robot vystřelí na bránu,
- střela z otočky - robot vystřelí otáčením se kolem vlastní osy,
- zastavení - robot se zastaví na aktuální pozici,
- přímo na bod - pošle robota na určité místo na hřišti - přímá cesta,
- navigace na bod - pošle robota na určité místo na hřišti - překážky v cestě,
- otočení na místě - robot se otočí na místě směrem k bodu,
- pohyb brankáře - brankář se pohybuje po brankovišti v závislosti na pozici míče.

Než se dostaneme k samotnému popisu dovedností je potřeba vysvětlit několik důležitých pojmů. Především jak takový pohyb vypadá a z čeho se skládá. V popisu vlastností robota bylo řečeno, že se pohybuje po dvou kolech, to mu umožňuje jednoduše manévrovat. Je-li popsán jen jako osa x a y pole, pak směr pohybu robota popisuje obrázek 20. Takže známe-li směr a souřadnice robota na hřišti, pak jsme schopni regulovat jeho pohyb na požadovanou souřadnici. Pomocí lineární rychlosti jsme schopni kontrolovat pohyb dopředu a dozadu. Naopak pomocí úhlové rychlosti zase ovlivňujeme jeho natáčení.
Vzorec reprezentující lineární rychlost:

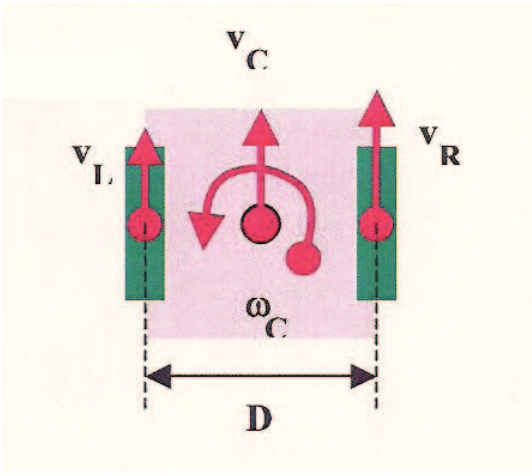
$$vc = \frac{vl + vr}{2}$$

Vzorec reprezentující úhlovou rychlost:

$$\omega_c = \frac{vl - vr}{D}$$



Obrázek 20: Souřadnicový systém robota [8]



Obrázek 21: Závislost mezi lineární a úhlovou rychlostí [8]

Konstanta	Hodnota
$obstacleArea$	15
$modificationArea$	10
$goalKeeperX$	5

Tabulka 9: Hodnoty konstant dovedností

Závislost mezi rychlostmi je vyobrazena na obrázku 21. V naší aplikaci budeme potřebovat provádět obě rychlosti současně. Budeme-li vykonávat nejdříve natočení a pak přesun, ztratíme čas, který je pro tuto hru velmi zásadní. Provedení natočení a přesun současně zabere méně času. Zároveň se robot neustále přibližuje ke svému cíli a je schopen podle toho také reagovat. Cílem je tedy propojit rychlosti.
Součet jednotlivých rychlostí, pro levé a pravé kolo:

$$V_L = K_p * d_e + K_a * \theta_e$$
$$V_R = K_p * d_e - K_a * \theta_e$$

Parametr d_e představuje vzdálenost robota od cílové pozice, nebo-li lineární rychlost. θ_e zase určuje úhel vychýlení od požadovaného směru. Jinak řečeno reprezentuje úhlovou rychlost, která ovlivňuje oblouk zatáčení. Konstanty K_p a K_a nejsou přesně určeny. Jedná se o modifikátory jednotlivých rychlostí.

Pro výpočet úhlu vychýlení od požadovaného směru použijeme, jako u vlastností metodu $atan2$. Vzdálenost robota od cíle získáme z Pythagorovy věty.

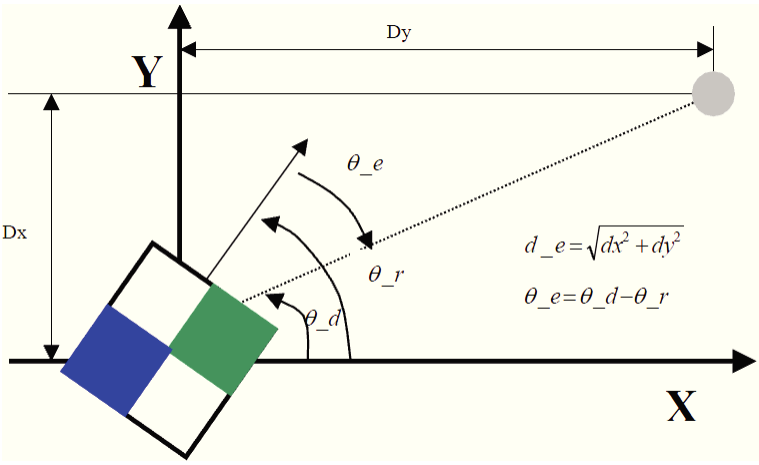
Tak jako ve vlastnostech i zde se vyskytují konstanty, které ovlivňují pohyb robotů. Pro přehlednost uvedu seznam s popisem.
Seznam konstant pro dovednosti:

- $obstacleArea$ - použití v dovednostech: Navigace na bod. Určuje oblast kolem překážek, která představuje zcela neproniknutelný prostor.
- $modificationArea$ - použití v dovednostech: Navigace na bod. Určuje oblast kolem překážky za prostorem „obstacleArea“, která představuje modifikační prostor.
- $goalKeeperX$ - použití v dovednostech: Pohyb brankáře. Určuje souřadnici x, po které se brankář před brankovištěm bude pohybovat.

Hodnoty konstant byly odvozeny z testů jednotlivých dovedností. Tabulka 9 zobrazuje hodnoty konstant. Nyní známe podstatné informace a víme jak funguje řízeni pohybu robota. Můžeme se tedy zaměřit na jednotlivé dovednosti.

6.1 Pozicování

V našem seznamu se pod tímto pojmenováním nenachází žádná dovednost. Jedná se o shluk několika dovedností, které si jsou velmi podobny a tvořit pro každou z nich metodu by nemělo příliš smysl. Jedná se především o tyto dovednosti:



Obrázek 22: Výpočet úhlu vychýlení a vzdáleností od cíle [8]

1. Jít za míčem.
2. Tah na bránu
3. Přímo na bod

Zamyslíme-li se, pak nám neunikne, že všechny metody mají společnou jednu vlastnost a to přemístit robota na nějakou pozici. Ať už se jedná o branku nebo míč, vždy půjde o cíl určený souřadnicí na hřišti. Dovednost Pozicování je právě ta, která zajišťuje přemístění robota do požadované pozice v nejkratším možném čase. To znamená, že robot na své cestě nemá žádnou překážku a je schopen se přemístit po přímé dráze. I to má však jistá omezení. Existují případy, kdy natočení robota a potom jeho přemístění bude mnohem efektivnější. Tento případ nastane, je-li robot oproti cílovému bodu příliš vychýlen. To znamená, že v první fázi by docházelo k oddalování od cíle a teprve až dosáhne požadovaného směru, začal by se přibližovat. Samozřejmě by to mohlo vést ke kolizím s mantinely a jiným problémům spojenými s závislosti lineární rychlosti na vzdálenosti od cíle. Tato dovednost je velmi podstatná, jelikož zabezpečuje nejrychlejší pohyb robotů po hřišti.

6.1.0.2 Algoritmus (výpis 24): Metoda obsahuje parametry souřadnic x a y. Díky tomu může posloužit více účelům, stačí nastavit souřadnice na cílový bod nebo objekt. V případě, že se jedná o vedení míče po hřišti, je potřeba nastavit parametr $hasBall$ na hodnotu $true$. Tím zajistíme, že robot se bude pohybovat takovým způsobem, aby během otáčení nepřišel o míč. Robot jej není schopen kontrolovat během otáčení na místě. Chceme-li robota někam přemístit musíme znát výpočet úhlu vychýlení a vzdálenost robota od cíle. Ten je vyobrazen na obrázku 22. Od cílové souřadnice odečteme souřadnice robota. takto získáme vzdálenosti mezi objekty na jednotlivých osách. Pomocí Pythagorovy věty spočítáme vzdálenost a funkcí $atan2$ úhel vychýlení. Vzdálenost je nutno

převést na centimetry, abychom počítali s lépe vnímanými jednotkami délky než jsou pixely. Nyní mohou nastat dvě možnosti:

- Úhel vychýlení je větší než 75°.
- Úhel vychýlení je menší než 75°.

Pro první případ to znamená, že nelze spojit lineární a úhlovou rychlost. Dráha robota by byla delší, než když provedeme nejdříve natočení a pak přiblížení. Zde je podstatná informace, zda vede míč. Není-li tomu tak, provede požadované natočení na místě. V opačném případě se otočí na malém poloměru, tak aby se neustále trochu pohyboval kupředu. Tak je zajištěno neustále tlačení míče a udržení kontroly nad ním.

V případě druhé možnosti úhel vychýleni je v rozsahu, kdy sloučení rychlosti povede k nejrychlejšímu přemístění na požadovanou pozici. Je však potřeba dodržet jistá kritéria. Vzdálenost robota od cíle určuje jeho rychlost, zatímco úhel vychýleni ovlivňuje úhlovou rychlost. V případě úhlové rychlosti rozdíl rychlostí mezi koly nesmí přesáhnout rozsahy z tabulky 7. V závislosti na vzdálenosti dochází k úpravě konstanty K_a tak, aby nedocházelo k nekontrolovatelnému pohybu. Se zmenšující vzdálenosti je robot schopen větší manévrovatelnosti.

6.2 Střela z otočky

Jedná se o jednu z možností provedení výstřelu na bránu. Střela není provedena přímým pohybem, ale díky otáčení robota na místě. Je-li míč dostatečně blízko a v takové pozici, aby touto dovedností nedošlo k ohrožení vlastní brány, pak provede výstřel. Za předpokladu, že detekce této dovednosti byla provedena dostatečně brzo, je zde dobrá šance na střelení gólu. Následný směr míče je zcela náhodný. Díky naší vlastnosti detekující, kdy je tuto možnost provést, jsme schopni střely alespoň vždy směřovat směrem na soupeřovu bránu. Dovednost má ještě jeden význam. Umožňuje brankáři vykopnutí míče, aniž by musel opustit brankoviště.

6.2.0.3 Algoritmus (výpis 10) :

```
void Skills::Spin(RobotSoccer& rs, Storage& st)
{
    if(st.Ball().Position().y >= rs.Position().y)
    {
        rs.VelocityLeft(20);
        rs.VelocityRight(-20);
    }
    else
    {
        rs.VelocityLeft(-20);
        rs.VelocityRight(20);
    }
}
```

Výpis 10: Dovednost střely z otočky

Dovednost sama o sobě není založena na příliš složitém výpočtu. Základem je určit pozici míče na ose y vůči pozici robota. To proto, aby robot se otáčel v takovém směru, že míč bude směřovat na bránu soupeře a nedojde k nechtěnému odkopu na naší polovinu nebo hůř střelení vlastního gólu. Je-li míč nad robotem respektive pod robotem bude se otáčet po směru hodinových ručiček respektive proti směru hodinových ručiček.

6.3 Navigace na bod

Tato dovednost představuje nejsložitější pohybování robotem. Během hry robot nebude mít nikdy příliš prostoru, aby se mohl pohybovat svobodně po hřišti. Zcela určitě ne bude-li mít ještě míč pod kontrolou a povede tah na bránu. Pro tyto případy vznikla dovednost Navigace na bod. Jejím cílem je pohybovat robotem takovým způsobem, aby došlo k objetí překážky. Robot provede objíždění po směru respektive proti směru hodinových ručiček v závislosti na cílové pozici vůči překážce. Dovednost končí obdobným výpočtem rychlostí jako Pozicování, ale s tím rozdílem, že zde dochází k výpočtu úhlu, pod jakým se překážka bude objíždět. Nachází-li se před ním překážka, mohou nastat tyto čtyři případy:

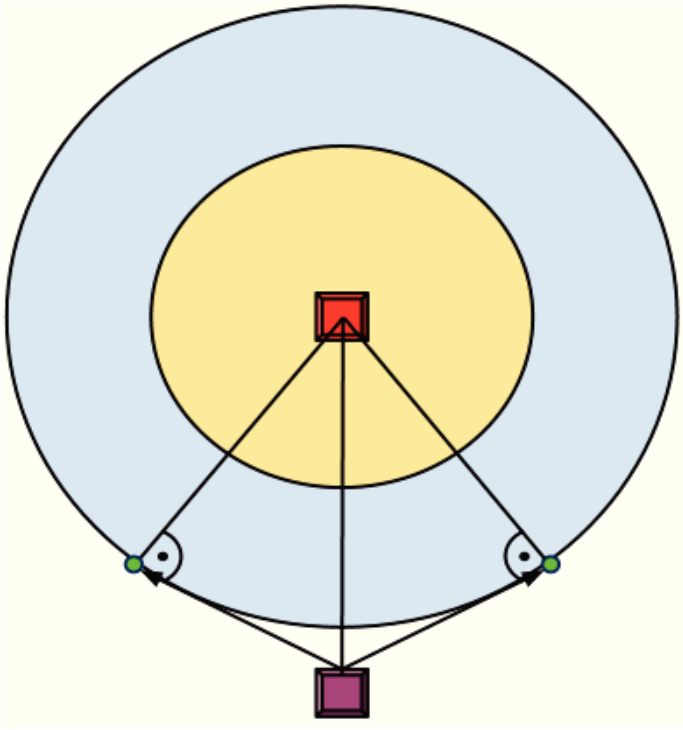
1. Cílová pozice je mimo nejbližší překážku, není potřeba objíždět.
2. Robot se nachází zcela mimo oblast překážky. Reprezentuje obrázek 23.
3. Robot se nachází v modifikační zóně překážky. Reprezentuje obrázek 24.
4. Robot se nachází v přímé zóně překážky. Reprezentuje obrázek 25.

První situace nastane, když nejbližší překážka nijak neomezuje našeho robota. Ten se pohybuje přímo na cíl. V případě druhé situace překážka se nenachází v dosahu našeho robota. To znamená, že má dostatek prostoru, aby rychle zareagoval a provedl objetí. V tomto případě robot je veden na pozice, jenž jsou zobrazeny na obrázku 23. Snahou je objíždět překážku po obvodu její zóny.

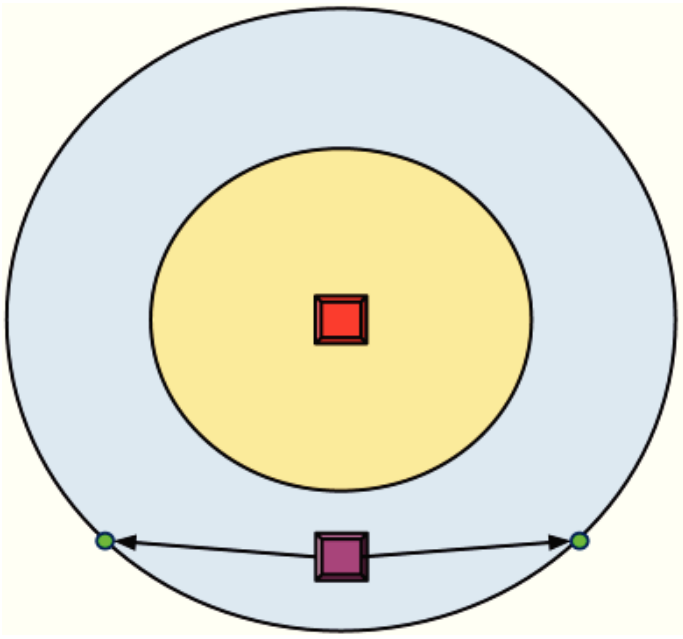
Ve třetí situaci se robot ocitne v modifikační zóně. Jeho následný pohyb závisí na tom, jak moc je blízko přímé zóně. To znamená, že nachází-li se dál od přímé zóny bude se snažit více objíždět než od překážky vzdalovat. Naopak čím bude této zóně blíže, bude se snažit od překážky vzdálit a úhel objetí se zmenší. Výsledná pozice pohybu je zobrazena na obrázku 24.

Čtvrtá situace představuje situaci, kdy robot se ocitne v přímé zóně. Takto není schopen překážku objet. Musíme předpokládat, že robot soupeře se během hry také pohybuje a tato zóna reprezentuje prostor jeho manévrování v jednom časovém úseku. Snahou je robota dostat z neprostřední blízkosti překážky, aby měl dostatek prostoru k provedení objetí. Výsledná pozice pohybu je zobrazena na obrázku 25.

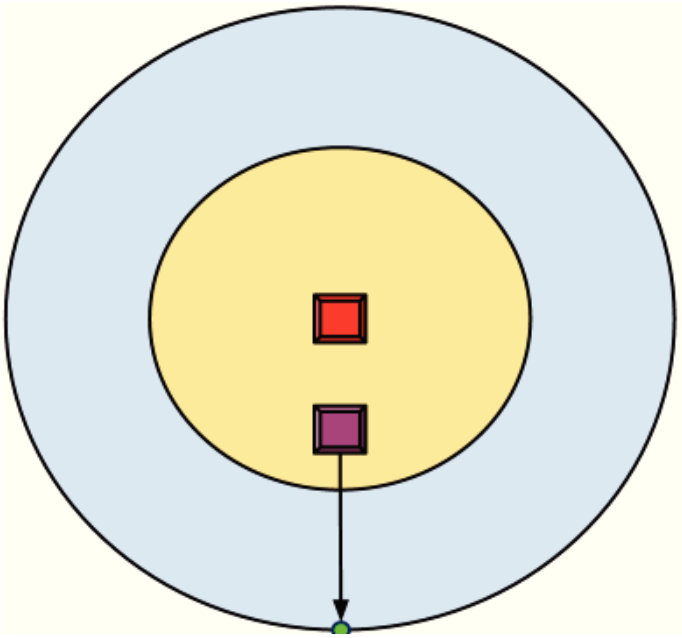
6.3.0.4 Algoritmus (výpis 25): Nejprve určíme nejbližší překážku na základě vzdáleností mezi objekty. Počítáme i s vlastními roboty. Jakmile překážka je určena odečteme od jejich souřadnic souřadnice robota. Ze vzniklých délek pro jednotlivé osy spočítáme



Obrázek 23: Robot mimo překážku



Obrázek 24: Robot uvnitř modifikační zóny



Obrázek 25: Robot uvnitř přímé zóny

pomocí funkce $atan2$ úhel $angle$, který představuje vychýleni robota oproti překážce. Od souřadnic cílového bodu odečteme souřadnice robota. Ze získaných délek pomocí funkce $atan2$ spočítáme úhel θ_d. Ten představuje vychýlení oproti cílovému bodu. Odečtením θ_d od $angle$ dostaneme úhel $diff\_angle$, který určuje směr objíždění překážky (po směru respektive proti směru hodinových ručiček). Je potřeba určit jak daleko se překážka nachází od přímé cesty robota k cíli. Délku získáme výpočtem: $lenght = (\text{vzdálenost mezi objekty}) * \sin(diff\_angle)$. Je-li $lenght$ menší než součet konstant $obstacleArea$ + $modificationArea$, pak překážka zasahuje do cesty a je nutno provést objíždění. V opačném případě výpočet rychlosti kol robota bude proveden jako u dovednosti Pozicování. Jakmile víme, že překážka zasahuje do cesty, určíme jak moc je blízko robotovi a podle toho zvolíme následující pohyb. Je-li vzdálenost menší než konstanta $obstacleArea$, pak robot se nachází v přímé zóně. Zde je cílem robota dostat mimo tuto zónu. Výsledný úhel pohybu θ_d dostaneme, když od $angle$ odečteme 180°. Výsledkem bude pohyb přesně na opačnou stranu, než je poloha překážky. Je-li vzdálenost menší než součet konstant $obstacleArea$ + $modificationArea$, pak robot se nachází v modifikační zóně. Zde je cílem robota dostat z blízkosti překážky, ale zároveň ji objet. Poměr mezi těmito směry je dán pozicí robota v této zóně.
Dočasné délky pro objíždění překážky po směru hodin:

$$tmp_x = (dist - obstacleArea) * \cos(angle + 1.5 * \pi)$$
$$tmp_y = (dist - obstacleArea) * \sin(angle + 1.5 * \pi)$$

Dočasné délky pro objíždění překážky proti směru hodin:

$$tmp_x = (dist - obstacleArea) * \cos(angle + 0.5 * \pi)$$
$$tmp_y = (dist - obstacleArea) * \sin(angle + 0.5 * \pi)$$

Dočasné délky pro oddalování od překážky:

$$tmp_x = (obstacleArea + modificationArea - dist) * \cos(angle + \pi)$$
$$tmp_y = (obstacleArea + modificationArea - dist) * \sin(angle + \pi)$$

Parametr $dist$ představuje vzdálenost mezi robotem a překážkou. Součtem dočasných délek pro výpočty oddalování a objíždění (po směru hodin nebo proti směru hodin) dostaneme průměrné délky. Na ty aplikujeme funkci $atan2$ a získáme výsledný úhel pohybu θ_d. Když vzdálenost neodpovídala ani jedné z předchozích podmínek, znamená to, že robot se nachází mimo dosah překážky. Robot sice není v dosahu, ale jeho cílový bod se nachází za překážkou. Má dostatek prostoru, aby mohl přímo objíždět a jeho směr povede na pozice znázorněné obrázkem 23. Podle vzorce:

$$\tan = \frac{a}{b},$$

spočítáme výsledný úhel pohybu θ_d. Stranu a nám představuje součet konstant $obstacleArea$ + $modificationArea$ a stranu b spočítáme z Pythagorovy věty jako:

$$b = dist^2 - (obstacleArea + modificationArea)^2$$

Výsledkem všech případů je úhel, pod kterým robot má směřovat na cílový bod. Jakmile jej máme spočítaný, pokračujeme stejným výpočtem pro určení rychlosti jako dovednost Pozicování.

6.4 Pohyb brankáře

Jedná se o jednu z nejdůležitějších dovedností. Můžeme mít ty nejlepší taktiky a dovednosti pro střelbu, ale se špatně se pohybujícím brankářem, nemáme mnoho šancí na úspěch. Dovednost je navržena tak, aby brankář od kritického bodu, začal kopírovat polohu míče. Snahou je neustále mu čelit a zamezit proniknutí za naší brankovou čáru. Brankář se pohybuje pouze na úrovní mezi horní a dolní tyčí brankoviště. Za předpokladu, že poloha míče je mimo brankoviště, bude vyčkávat u tyče. Pokryje tak prostor, pro případnou střelu z boku. Na ose x se pohybuje po konstantní vzdálenosti od brankoviště. Tuto vzdálenost určuje konstanta $goalKeeperX$.

6.4.0.5 Algoritmus (výpis 26): Nejprve musíme určit x souřadnici, po níž se brankář bude pohybovat. Konstantu $goalKeeperX$ pomocí metody XY_to_Px převedeme na pixely. K souřadnici centra naší brány přičteme hodnotu konstanty. Výsledkem je souřadnice x. Metoda obsahuje parametr $center$, kterým říkáme kam se má robot posunout. Nastavíme-li jeho hodnotu na $true$, znamená to, že robot přesune do výchozí polohy, tj.

středu brány. Jinak bude kopírovat polohu míče. Souřadnici y nastavíme podle y souřadnice míče. Je-li tato souřadnice menši respektive větší než souřadnice dolní respektive horní tyče, dojde k nastavení y podle tyčí. Robot nepojede dál, než jsou tyče brány. Od souřadnic x,y odečteme souřadnice brankáře. Ze získaných délek pro jednotlivé osy pomocí Pythagorovy věty spočítáme vzdálenost a funkcí $atan2$ úhel brankáře vůči požadované pozici. Když vzdálenost je menší než půl centimetru dochází už k zastavení robota. Výpočet pro brankáře je veden vůči jeho středu. To znamená, že analýza jej detekuje přesně u tyče, ale svou šířkou přesahuje, proto jej zastavujeme o něco dříve. Jako jediný je navržen tak, aby byl schopen jezdit i pozpátku. Takto dosáhneme dostatečné flexibility pokrytí brány. Od úhlu vychýlení odečteme úhel rotace brankáře. Je-li výsledek v rozsahu -5 až 5, pak robot se bude pohybovat směrem nahoru v přímém směru. Naopak je-li výsledek v rozsahu -175 až -180 nebo 175 až 180, pak robot se bude pohybovat směrem dolu. Nenáleží-li výsledek ani do jedno z rozsahu, znamená to, že robot je vychýlen. Může být způsobeno kolizí se soupeřem atd. V tomto případě robot se přesune na požadovanou pozici.

6.5 Střela

Tato dovednost jako jediná realizuje střelu na bránu v přímém směru. Robot se pokusí dosáhnout vysoké rychlosti, aby střela měla šanci ohrozit branku soupeře. Naše vlastnosti pro určení možnosti střely počítají s výpočtem na střed brány. Jelikož konstanty jsou navrženy na malé odchylky v úhlech, střela bude vždy směřovat na bránu. Odchylka zde představuje vzdálenost od centra, kde míč protne brankovou čáru.

6.5.0.6 Algoritmus (výpis 27): Snahou je míč ještě před výstřelem směřovat na střed brány. Od souřadnic míče odečteme souřadnice robota. Ze vzniklých délek pomocí funkce $atan2$ spočítáme úhel mezi robotem a míčem. Odečteme-li od tohoto úhlu rotaci robota, získáme úhel vychýlení robota od kurzu k míči. Tento úhel částečně ještě upravuje směr střely. Metoda obsahuje parametr $fast$. Určuje zda se má jednat o rychlou ($false$) nebo prudkou ($true$) střelu. Rychlá střela se využije pro predikované střely, aby nedošlo k minutí míče. Prudká střela zase na přímou střelu, kdy víme, že míč se opravdu nachází před robotem.

6.6 Zastavení

Cíl dovednosti je už jasný z jejího nazvu. Nedělá nic jiného, než že nastaví rychlosti kol robota na 0 a dojde k jeho zastavení. Metoda je zobrazena ve výpisu 11

```
void Skills::Stop(RobotSoccer& rs)
{
    rs.VelocityLeft(0);
    rs.VelocityRight(0);
}
```

Výpis 11: Dovednost pro zastavení robota

6.7 Otočení na místě

Dovednost upravuje rotaci robota na místě. Obzvláště v defenzivní pozici by mohlo být potřeba, aby se robot natáčel neustále na míč a podobně.

6.7.0.7 Algoritmus (výpis 28): Metoda obsahuje parametry pro souřadnice x, y. Od těchto souřadnic odečteme souřadnice robota. Ze vzniklých délek pomocí funkce $atan2$ spočítáme úhel mezi robotem a bodem, na který se má natočit. Od tohoto úhlu odečteme rotaci robota a výsledek představuje vychýlení od požadovaného směru. Podle toho zda hodnota bude kladná nebo záporná, bude zvolen směr natáčení.

6.8 Souhrn dovedností

Cílem bylo vytvořit takové dovednosti, aby robot se mohl aktivně zapojovat do hry za všech okolností, které během hry nastanou. Robot je nyní schopen pohybovat se po nejkratší přímé dráze na zvolený bod, za předpokladu, že není blokován překážkou. Nemůžeme počítat s neustále volným prostorem pro robota. To dalo podnět ke vzniku dovednosti pro objíždění překážek. Robot je tedy schopen vyhnout se protivníkovi. Brankář je řízen dovedností navrženou přímo pro něj a je schopen čelit blížícímu se míči. Dochází takto k pokrytí brankoviště. Jako jediný je schopen pohybovat se i pozpátku pro dosažení větší flexibility, při změně směru. Brankář není nucen opouštět brankoviště díky dovednosti střely z otočky. Je schopen odkopnout míč, jakmile je v jeho dosahu. tato dovednost slouží také ke střelbě na bránu soupeře. Je-li provedena rychle, brankář soupeře nebude mít mnoho času na reakci. Když už se bavíme o střelbě, máme dovednost jenž zajišťuje dostatečnou rychlost robotovi, aby provedl přímou střelu. Můžeme ovlivnit nastavením parametru jak moc rychle to má provést. Později byly navrženy dovednosti pro zastavení a otočení na místě. Ty jako poslední uzavírají základní dovednosti, jenž roboti musí být schopni vykonat, aby se mohli efektivně pohybovat na hřišti.

7 Závěr

Cílem práce bylo vytvořit vlastní třídu, která by se staralo o zasílání řídících signálu jednotlivým robotům. Tím jsme se mohli odpoutat od doposud používané aplikace a zaměřit se na vývoj vlastní. Takto jsme nebyli ničím omezování a nemuseli jsme se přizpůsobovat. V týmu jsme společnými silami vyvíjeli jednotlivé části zcela nezávisle na předešlých výzkumech.

Když jsem obdržel roboty, byly to pro mě jenom malé kostičky, které mi samy o sobě příliš neprozradily. Nebyly k ním žádné informace ani manuál o jejich dovednostech. Ačkoli robot působí velmi jednoduše, jeho pohybové možnosti jsou opravdu obrovské. Jelikož je navržen, tak že jeho pohyb zprostředkovávají dva zcela nezávislé motory, které se starají o rychlost otáčení levého a pravého kola, je schopen dosáhnout rychlého zrychlení a především manévrovacích schopností na malém prostoru. Tyto poznatky a mnoho dalších informací, bylo získáno z prováděných testů. Snahou bylo vytvořit takové experimenty, abychom maximálně pochopili pohybové dovednosti robota. Všechny poznatky jsem zaznamenal do této práce a jsou uvedeny ve třetí kapitole. Tyto informace se však stahují pouze na jeden typ robota. Rozhodne-li se univerzita pro nákup nových robotů, pak jednotlivé testy je potřeba provést znovu, abychom měli pravdivá data. Mnoho výpočtů je založených na těchto poznatcích.

Jakmile jsem dosáhl dostatečného poznání možností robotů, bylo potřeba zpracovat vlastnosti. Vlastnosti představují podstatné rozhodovací body volby akce robota. Jsou základním prvkem pro tvorbu taktik. Vlastnost vychází z reálných dat objektů, které se nacházejí na hřišti. Jejich cílem je dostatečně popsat situaci na hřišti pro všechny objekty a jednotlivé roboty. Zde bylo cílem dosáhnout takového pokrytí, aby na každou situaci byl robot schopen reagovat a aktivně se zapojovat do hry.

Vlastnosti samy o sobe nemají význam, jelikož nekontroluji přímo pohyb robota, ale jen volbu jeho akce. Nastavování rychlostí jednotlivých kol zajišťují dovednosti, které jsou volány z taktik. Každá dovednost se zaměřuje na specifický pohyb robota, který v dané situaci je potřeba vykonat. Zde bylo také cílem vytvořit dostatečný soubor dovedností, aby na každou detekovanou vlastnost byl robot schopen reagovat pohybem.

Závěrem bych chtěl říci, že práce na tomto projektu byla velmi zajímavá. Především spolupráce v malém týmu byla opět zkušeností do života. Po celou dobu jsme se scházeli v pravidelných termínech a vzájemně se informovali o svých postupech a problémech. Ty jsme společnými silami rozebírali a navrhovali optimální řešení. Fotbal robotů je opravdu fascinujícím projektem, kde se člověk ocitne v kontaktu s mnoha vědeckými disciplínami.

7.1 Zlepšení do budoucna

Během čtení nejrůznějších materiálu k robotům jsem v jednom článku viděl zajímavě zpracovaný design robotů. Tito roboti nebyli upraveni pouze zepředu ke hře, ale také zezadu. To znamená, že byli schopni vést míč na obě strany. Jak víme, robot nemá rozlišné vlastnosti pro směry. Chová se stejně ať jede kupředu nebo pozpátku. To zajišťuje rychlejší pohyblivost na hřišti. Tento robot má mnohem větší úhel, ve kterém se pohybuje přímo, aniž by nejdříve prováděl natočení a teprve pak přiblížení. Jenom dojde ke změně

znamének před hodnotami rychlosti, aby robot směřoval pozpátku na svůj cíl. Bude-li katedra uvažovat o pořízení nových robotů, doporučuji poohlédnout se právě po tomto modelu.

Během testů byl zaznamenán jeden velký nedostatek. Neexistoval žádný výpočet rychlosti míče, proto se s tímto parametrem nepracovalo. Jedná se však o velmi podstatnou záležitost. Když se robot měl pohybovat za míčem, udržoval s ním konstantní rychlost. Tedy místo, aby jej měl pod kontrolou, pouze jel za ním. Tento problém by řešila detekce rychlosti míče. Jakmile bychom k výpočtům jednotlivých dovedností jako jsou: Pozicování a Navigace na bod, přičetli tuto rychlost, robot by míč nepronásledoval, ale dostal ho pod kontrolu.

8 Reference

[1] Awang Hendrianto Pratomo, Anton Satria Prabuwono, Mohd. Shanudin Zakaria, Khairuddin Omar, Md. Jan Nordin, Shahnorbanun Sahran, Siti Norul Huda Sheikh Abdullah and Anton Heryanto, *Position and Obstacle Avoidance Algorithm in Robot Soccer,* Malaysia 2010.

[2] Claude, Sammut, *Robot Soccer*.

[3] Daniel Štrba, *Architektura hry fotbalu robotů*,Ostrava 2011, Diplomová práce.

[4] Dr. Mannes Poel, Dr. Albert Schoute, Dr. Job Zwiers, *Artificial Intelligence in a multi agent robot soccer domain*,The Netherlands, August 2003.

[5] FIRA, *http://www.fira.net/*.

[6] Knihovna math.h, *http://www.sallyx.org/sally/c/c21.php*.

[7] Matematická funkcia - ATAN2, *http://wiki.openoffice.cz/wiki/Matematick%C3%A1_funkcia_-_ATAN2*.

[8] Yujin Robotics Co.,Ltd, *Robot Soccer YSR-A System Manual,* Soul Korea.

9 Příloha

Zdrojové kódy nalezneme také na přiloženém CD k této diplomové práci.

```
void TacticChooser::chooseTactic(Storage& storage)
{
    RobotFeatures* rf = new RobotFeatures();
    GlobalFeatures* gf = new GlobalFeatures();
    comm = new Comm();
    gs = new GameSetting();
    gf->SetGlobalFeatures(storage);
    Skills * sk = new Skills();
    velocityArray = new int [gs->NUMBER_OF_ROBOTS()];
    int arraySlot = 0;

    for(int i=0;i<gs->NUMBER_OF_ROBOTS();i++)
    {
        RobotSoccer rs = storage.MyRobots()[i];
        switch(rs.GetRole())
        {
        case RobotSoccer::Role::utocnik:
            if (gf->GetMicNaJejichStrane())
            {
                //  strela  na presny mic
                if ( rf->CanShot(rs,storage))
                {
                    sk->Shoot(rs,storage,true);
                }
                // strela  na predikovany mic
                else if  ( rf->ShotOpportunity(rs,storage,true))
                {
                    sk->Shoot(rs,storage,false);
                }
                // strela  z  otocky
                else if ( rf->ShotSpin(rs,storage))
                {
                    sk->Spin(rs,storage);
                }
                // strela  na predikovany mic z daly
                else if  ( rf->PredictedShot(rs,storage) && rf->FreeSpace(rs,storage,storage.Ball().
                    Position().x,
                    storage.Ball().Position().y,false)) // free  na souradnice k mici
                {
                    sk->Shoot(rs,storage,false);
                }
                //  strela  od mantaku
                else if ( rf->BiliardScoring(rs,storage))
                {
                    sk->Shoot(rs,storage,true);
                }
                //ma mic a snazi se dostat k brane
```

```
        else if ( rf->BallUnderControl(rs,storage))
        {
            //Ma cisty prostor jede primo, jinak objizdi nejblizsi prekazku
            if ( rf->FreeSpace(rs,storage,gs->GoalOppCentreX(),gs->GoalOppCentreY(),
                 true)) //free na souradnice brany
            {
               sk->Position(rs,gs->GoalOppCentreX(),gs->GoalOppCentreY(),true);
            }
            else
            {
               sk->ObstacleAvoidance(rs,storage,gs->GoalOppCentreX(),gs->
                    GoalOppCentreY(),true);
            }
        }
        //nema mic a snazi se k nemu dostat
        else
        {
            //Ma cisty prostor jede primo, jinak objizdi prekazku
            if ( rf->FreeSpace(rs,storage,storage.Ball().Position().x,storage.Ball().Position().y
                 ,false)) //free na souradnice k mici
            {
               sk->Position(rs,storage.Ball().Position().x,storage.Ball().Position().y,false);
            }
            else
            {
               sk->ObstacleAvoidance(rs,storage,storage.Ball().Position().x,storage.Ball().
                    Position().y,false);
            }
        }
    }
    // kdyz je mic na nasi strane utocnik je rizen strategii
    else if (gf->GetMicNaNasiStrane())
    {
        // ma-li k pozici strategie volny prostor, jede primo, jinak objizdi
        if ( rf->FreeSpace(rs,storage,rs.PositionMove().x,rs.PositionMove().y,false))
        {
            sk->Position(rs,rs.PositionMove().x,rs.PositionMove().y,false);
        }
        else
        {
            sk->ObstacleAvoidance(rs,storage,rs.PositionMove().x,rs.PositionMove().y,false);
        }
    }
    break;
case RobotSoccer::Role::obrance:
    // kdyz je mic na jejich polovine obrance je rizen strategii
    if (gf->GetMicNaJejichStrane())
    {
        // ma-li k pozici strategie volny prostor, jede primo, jinak objizdi
        if ( rf->FreeSpace(rs,storage,rs.PositionMove().x,rs.PositionMove().y,false))
        {
            sk->Position(rs,rs.PositionMove().x,rs.PositionMove().y,false);
        }
        else
```

```
        {
            sk->ObstacleAvoidance(rs,storage,rs.PositionMove().x,rs.PositionMove().y,false);
        }
    }
    else if(gf->GetMicNaNasiStrane())
    {   // kdyz je mic za kritickou hranici farDistance je primarnim cilem mic vykopnout
        //goalcentreX je zaporna hodnota && konstanta farDistance kladna - proto pred
            kosntantu vkladam"-"
        if(rs.Position().x < (gs->GoalHomeCentreX() - (-rf->XY_to_Px(farDistance)))
            && rf->KickAway(rs,storage))
        {
            sk->Shoot(rs,storage,true);
        }
        // mic jeste neprekrocil kriticou linii - mali mic pod kontrolou bude se snazit
            vyrazit na branu soupere
        else if(rf->BallUnderControl(rs,storage))
        {   // ma volnou cestu na branu - jede primo
            if(rf->FreeSpace(rs,storage,gs->GoalOppCentreX(),gs->GoalOppCentreY(),
                true)) // free na souradnice brany
            {
                sk->Position(rs,gs->GoalOppCentreX(),gs->GoalOppCentreY(),true);
            }
            else
            {
                sk->ObstacleAvoidance(rs,storage,gs->GoalOppCentreX(),gs->
                    GoalOppCentreY(),true);
            }
        }
        else
        {
            if(rf->FreeSpace(rs,storage,storage.Ball().Position().x,storage.Ball().Position().y
                ,false))
            {
                sk->Position(rs,storage.Ball().Position().x,storage.Ball().Position().y,false);
            }
            else
            {
                sk->ObstacleAvoidance(rs,storage,storage.Ball().Position().x,storage.Ball().
                    Position().y,false);
            }
        }
    }
    break;
case RobotSoccer::Role::brankar:
    // mic je blisko brankare - ten se zacne otacet aby ho vykopnul
    if(rf->ShoudSpin(rs,storage))
    {
        sk->Spin(rs,storage);
    }
    // mic blizko brany - brankar kopiruje y souradnici mice
    else if (rf->GoalMove(rs,storage))
    {
        sk->GoalKeeperPosition(rs,storage,false);
    }
```

```
            else
            {
                // mic je daleko od brany - brankar vyckava ve stredu brany
                sk->GoalKeeperPosition(rs,storage,true);
            }
            break;
        case RobotSoccer::Role::nedefinovany:
            // ma-li k pozici strategie volny prostor, jede primo, jinak objizdi
            if ( rf->FreeSpace(rs,storage,rs.PositionMove().x,rs.PositionMove().y,false))
            {
                sk->Position(rs,rs.PositionMove().x,rs.PositionMove().y,false);
            }
            else
            {
                sk->ObstacleAvoidance(rs,storage,rs.PositionMove().x,rs.PositionMove().y,false);
            }
            break;
        }
        velocityArray[arraySlot] = rs.VelocityLeft();
        arraySlot++;
        velocityArray[arraySlot] = rs.VelocityRight();
        arraySlot++;
    }
    comm->VehicleSpeed(velocityArray);
}
```

Výpis 12: Výběr taktiky

```
void GlobalFeatures::SetGlobalFeatures(Storage st)
{
    int ballX = st.Ball().Position().x;
    int ballY= st.Ball().Position().y;
    int ballRotation = 0;
    int branaX= gs->GoalHomeCentreX();
    int branaY= gs->GoalHomeCentreY();
    int goalPostUX= gs->GoalHomePostUpX();
    int goalPostUY= gs->GoalHomePostUpY();
    int goalPostDX= gs->GoalHomePostDoX();
    int goalPostDY= gs->GoalHomePostDoY();
    int tempAngle1,tempAngle2;
    double dx,dy,de;

    //Vypocet Tlaku mice
    double smerMice=0;
    if (ballRotation > 90 && ballRotation < 270)
    {
        if (ballRotation <= 180)
            smerMice = (ballRotation - 90) * (20/90); //20/90 protoze nas mic ohrozuje /respektive
                smeruje na nasi stranu jen ve dvou kvadrantech
        else
            smerMice = (270 - ballRotation) * (20/90);
    }
    double vzdalenostOdBrany=0;
    if (ballX < 0)
```

```
    {
        dx = branaX − ballX;
        dy = branaY − ballY;
        vzdalenostOdBrany = (sqrt(dx∗dx + dy∗dy))∗(20/(abs(branaX)∗2)); // 20/branaX∗2 urcuje kolik
            je procent jeden pixel na cele ose X
    }
    double souradniceX = (branaX − ballX) ∗ (60/((abs(branaX))∗2)); //rozdeleni 60\% mezi hraci
        ploxhu na ose X. Brany jsou od stredu stejne vzdalene(proto ∗2)

    if ((smerMice + vzdalenostOdBrany + souradniceX) >= 60) tlakMice = true;
    else tlakMice = false;

    //Mic na nasi strane?
    if (ballX < 0) micNaNasiStrane = true;
    else micNaNasiStrane = false;

    //Mic na jejich strane?
    if (ballX >= 0) micNaJejichStrane = true;
    else micNaJejichStrane = false;

    //Pohyb na nasi polovinu?
    if (ballRotation > 90 || ballRotation < −90) pohybNaNasiPolovinu = true;
    else pohybNaNasiPolovinu = false;

    //Mic u nasi branky
    dx = branaX − ballX;
    dy = branaY − ballY;
    de = sqrt(dx∗dx + dy∗dy);
    de = rf−>XY_to_Cm(de);
    if (de < closeDistance) micUnasiBranky = true;
    else micUnasiBranky = false;

    //Pohyb do nasi brany
    dx = goalPostUX − ballX;
    dy = goalPostUY − ballY;
    tempAngle1 = (int)(180/3.14 ∗ atan2(dy,dx)); // vypocet uhlu k horni tyci

    dx = goalPostDX − ballX;
    dy = goalPostDY − ballY;
    tempAngle2 = (int)(180/3.14 ∗ atan2(dy,dx)); //vypocet k dolni tyci

    if (ballRotation <= tempAngle1 && ballRotation >= tempAngle2) pohybDoNasiBrany = true;
    // na nasi strane pocitame s uhly kolem 180 => komplikace kdyz horni je zaporna a dolni
        kladna
    else if (tempAngle1 < 0 && tempAngle2 > 0)
    {
        if ((ballRotation <= tempAngle1 && −ballRotation >= tempAngle2) ||
            (−ballRotation <= tempAngle1 && ballRotation >= tempAngle2)) pohybDoNasiBrany =
                true;
        else pohybDoNasiBrany = false;
    }
    else pohybDoNasiBrany = false;
}
```

Výpis 13: Výpočet globálních vlastností

```
bool RobotFeatures::BallUnderControl(RobotSoccer& rs, Storage& st)
{
    // temp veci
    int robotX =rs.Position().x;
    int robotY =rs.Position().y;
    int ballX =st.Ball().Position().x;
    int ballY =st.Ball().Position().y;
    int robotRotation = rs.Rotation();
    //zustanou
    double dx;
    double dy;
    double ballDistance;
    int ballAngle;

    dx = ballX - robotX;
    dy = ballY - robotY;
    ballDistance = sqrt(dx * dx + dy * dy);
    ballDistance = XY_to_Cm(ballDistance);

    ballAngle = (int)(180/M_PI * atan2(dy,dx));
    ballAngle = robotRotation - ballAngle;      // je potreba urcit uhel podle natoceni robota

    while(ballAngle > 180) ballAngle -= 360;
    while(ballAngle < -180) ballAngle += 360;

    if (ballDistance < possessionRange && ballAngle <= maxControledAngle && ballAngle >= -
        maxControledAngle) return true;
    else return false;
}
```

Výpis 14: Výpočet pro vlastnost Má míč

```
bool RobotFeatures::CanShot(RobotSoccer& rs, Storage& st)
{
    // temp veci
    int robotX =rs.Position().x;
    int robotY =rs.Position().y;
    int robotRotation = rs.Rotation();
    int ballX = st.Ball().Position().x;
    int ballY =st.Ball().Position().y;
    int brankaX= gs->GoalOppCentreX();
    int brankaY = gs->GoalOppCentreY();
    //zustanou
    double dx;
    double dy;
    double ballDistance;
    int ballAngle;
    int goalAngle;
    int tempAngle;
```

```
    //souradnice centra brany
    dx = brankaX − robotX;
    dy = brankaY − robotY;

    goalAngle = (int)(180/M_PI ∗ atan2(dy,dx));
    //vysledny uhel natoceni robota k brane
    tempAngle = goalAngle − robotRotation;
    while (tempAngle > 180) tempAngle −= 360;
    while (tempAngle < −180) tempAngle += 360;
    if (tempAngle <= maxGoalieAngle && tempAngle >= −maxGoalieAngle)
    {
        //souradnice mice
        dx = ballX − robotX;
        dy = ballY − robotY;

        ballAngle = (int)(180/M_PI ∗ atan2(dy,dx));
        ballDistance = sqrt(dx ∗ dx + dy ∗ dy);        //  vzdalenost mice od robota
        ballDistance = XY_to_Cm(ballDistance);     //  prevod Px na Cm

        if (ballDistance <= maxRadius)
        {
            tempAngle = goalAngle − ballAngle;
            while (tempAngle > 180) tempAngle −= 360;
            while (tempAngle < −180) tempAngle += 360;
            if (tempAngle <= maxAngle && tempAngle >= −maxAngle) return true;
            else return false;
        }
        else return false;
    }
    else return false;
}
```

Výpis 15: Výpočet pro vlastnost Může vystřelit

```
bool RobotFeatures::FreeSpace(RobotSoccer& rs,Storage& st, int pointX, int pointY, bool
    goalCentre)
{
    //temp veci
    int robotX = rs.Position().x;
    int robotY = rs.Position().y;
    int robot2X;
    int robot2Y;
    //zustanou
    bool result = false;
    double b = 0;
    double alfa = 0;
    int angleCentre = 0;
    int angleOponent = 0;
    double dx = 0;
    double dy = 0;
    double vc = 0;
    double distance = 0;

    for(int i=0;i<gs->NUMBER_OF_ROBOTS();i++)
```

```
    {
        for(int j=0;j<2;j++)
        {
            if(j=0)
            {
                robot2X = st.MyRobots()[i].Position().x;
                robot2Y = st.MyRobots()[i].Position().y;
                if(robot2X==robotX && robot2Y==robotY) continue;
            }
            else if(j=1)
            {
                robot2X = st.OppRobots()[i].Position().x;
                robot2Y = st.OppRobots()[i].Position().y;
            }
            dx = robot2X - pointX;
            dy = robot2Y - pointY;
            distance = sqrt(dx * dx + dy * dy);
            distance = XY_to_Cm(distance);   //prevod Px na Cm
            //snaha udelat to vseobecne pro jakykoiv bod
            if(distance < maxGoalieRadius && goalCentre)
                result = true;
            else if((robot2X > robotX && robot2X < pointX ) || (robot2X > pointX && robot2X <
                robotX))
            {
                dx = robotX - pointX;
                dy = robotY - pointY;
                angleCentre = (int)(180/M_PI * atan2(dy,dx));

                dx = robotX - robot2X;
                dy = robotY - robot2Y;
                b = sqrt(dx * dx + dy * dy);
                angleOponent = (int)(180/M_PI * atan2(dy,dx));

                alfa = angleCentre - angleOponent;
                while(alfa >180 ) alfa = alfa - 360;
                while(alfa < -180) alfa = alfa + 360;
                alfa = abs(alfa);
                alfa = (alfa*M_PI)/180;
                vc = b * sin(alfa);
                vc = XY_to_Cm(vc);   //prevod Px na Cm

                if(vc < pathWidth)
                {
                    result = false;  //  musime pripocitat sirku  robota(5 + 3,8)
                    return result;
                }
                else result = true;
            }
            else result = true;
        }
    }
    return result;
}
```

Výpis 16: Výpočet pro vlastnost Volný prostor na bránu

```
bool RobotFeatures::ShotOpportunity(RobotSoccer& rs, Storage& st, bool our)
{
    // temp veci
    int robotX =rs.Position().x;
    int robotY =rs.Position().y;
    int robotRotation = rs.Rotation();
    int ballPredictedX = st.Ball().PredictedPosition().x;
    int ballPredictedY = st.Ball().PredictedPosition().y;
    int brankaX, brankaY;
    //bude potreba resit pro nasi a souperovu branu podle toho ktery robot se bude propocitavat H/
        O
    if (our)
    {
        brankaX = gs->GoalHomeCentreX();
        brankaY = gs->GoalHomeCentreY();
    }
    else
    {
        brankaX = gs->GoalOppCentreX();
        brankaY = gs->GoalOppCentreY();
    }

    //zustanou
    double dx;
    double dy;
    double pX;
    double pY;
    double pRadius;
    double alfa,tempAngle;

    dx = brankaX - robotX;
    dy = brankaY - robotY;
    tempAngle = (int)(180/3.14 * atan2(dy,dx));
    alfa = tempAngle - robotRotation;
    while(alfa > 180) alfa -= 360;
    while(alfa < -180) alfa += 360;

    if (alfa < maxGoalieAngle && alfa > -maxGoalieAngle) // testujeme zda robot je natoceny ve
        smeru na branu
    {
        alfa = abs(tempAngle);
        if (alfa > 90) // zajistim stupne do 90 podle kvadrantu -/+ nema vyznam pro vypocet x/y
        {
            alfa = abs(alfa - 180);
        }
        alfa = (alfa*M_PI)/180; // prevod na radiany
        dx = cos(alfa) * XY_to_Px(runUp); // delku prepony prevadime z Cm na Px
        dy = sin(alfa) * XY_to_Px(runUp);
        // vypocet Y souradnice P bodu
        if (tempAngle >= 0) pY = robotY + dy;
```

```
        else pY = robotY − dy;
        // vypocet X souradnice P bodu
        if (brankaX > 0) pX = robotX + dx;
        else pX = robotX − dx;

        //vypocet radiusu kolem bodu P
        dx = pX − ballPredictedX;
        dy = pY − ballPredictedY;
        pRadius = sqrt(dx∗dx + dy∗dy);
        pRadius = XY_to_Cm(pRadius);
        if (pRadius < radius) return true;
        else return false;
    }
    else return false;
}
```

Výpis 17: Výpočet pro vlastnost Pravděpodobnost vystřelení

```
bool RobotFeatures::BiliardScoring(RobotSoccer& rs, Storage& st)
{
    // temp veci
    int robotX = rs.Position().x;
    int robotY = rs.Position().y;
    int robotRotation = rs.Rotation();
    int ballX = st.Ball().Position().x;
    int ballY = st.Ball().Position().y;
    int brankaHX = gs->GoalOppCentreX();
    int brankaHY = gs->GoalOppCentreY();
    int gameFiledYP = gs->GameFieldDoY();
    int gameFieldYM = gs->GameFieldUpY();
    //zustanou
    double dx;
    double dy;
    double pX;
    double pY;
    double a,b;
    double pRadius;
    double alfa,tempAngle,temp;

    if (robotX > 0 && robotX < (brankaHX − maxGoalieRadius))
    {
        dx = robotX − brankaHX;
        b = (int)dx/2;
        if (ballY > 0)
        {
            a = gameFiledYP − brankaHY; // zjistujeme delku strany a
            dy = gameFiledYP − robotY;
        }
        else
        {
            a = brankaHY − gameFieldYM; // zjistujeme delku strany a − prehozeny vypocet abychom
                dostali kladnou hodotu
            dy = robotY − gameFieldYM;
            robotRotation −= 360; // kdyz dostanu uhel 299 tak mne zajima zbytek do 360
```

```
        }
        alfa = (int)(180/M_PI * atan2(a,b));

        dx = b;
        tempAngle = (int)(180/M_PI * atan2(dy,dx)); //uhle u robota
        alfa = alfa - tempAngle;
        temp = tempAngle - robotRotation;
        while(temp > 180) temp -= 360;
        while(temp < -180) temp += 360;

        if (temp < maxBiliardRobotAngle && temp > -maxBiliardRobotAngle)//urcuje jestli robot je
            natoceny tim smerem
        {
            if (alfa < maxBiliardAlfaAngle && alfa > -maxBiliardAlfaAngle)
            {
                tempAngle = (tempAngle*M_PI)/180;
                dx = cos(tempAngle) * XY_to_Px(runUp); // delku prepony prevadime z Cm na Px
                dy = sin(tempAngle) * XY_to_Px(runUp);
                // vypocet Y souradnice P bodu
                if (ballY > 0) pY = robotY + dy;
                else pY = robotY - dy;
                // vypocet X souradnice P bodu
                pX = robotX + dx;
                //vypocet radiusu kolem bodu P
                dx = pX - ballX;
                dy = pY - ballY;
                pRadius = sqrt(dx*dx + dy*dy);
                pRadius = XY_to_Cm(pRadius);
                if (pRadius < radius) return true;
                else return false;
            }
            else return false;
        }
        else return false;
    }
    else return false;
}
```

Výpis 18: Výpočet pro vlastnost Mantinel

```
bool RobotFeatures::ShoudSpin(RobotSoccer& rs, Storage& st)
{
    // temp veci
    int robotX = rs.Position().x;
    int robotY = rs.Position().y;
    int ballX = st.Ball().PredictedPosition().x;
    int ballY = st.Ball().PredictedPosition().y;
    int goalCentreX = gs->GoalHomeCentreX();
    //zustanou
    double dx;
    double dy;
    double de;
    if (goalCentreX < ballX && robotX > ballX)
        return false;
```

```
    else
    {
        dx = robotX − ballX;
        dy = robotY − ballY;
        de = sqrt(dx∗dx + dy∗dy); // vzdalenost mice od robota
        de = XY_to_Cm(de);
        if (de < maximumHitDistance) return true;
        else return false;
    }
}
```

Výpis 19: Výpočet pro vlastnost Otáčení

```
bool RobotFeatures::ShotSpin(RobotSoccer& rs, Storage& st)
{
    double dx,dy,de,angle,dist;
    double predictedBallX = st.Ball().PredictedPosition().x;
    double predictedBallY = st.Ball().PredictedPosition().y;
    double brankaX = gs−>GoalOppCentreX();
    double goalPostUX = gs−>GoalOppPostUpX();
    double goalPostUY = gs−>GoalOppPostUpY();
    double goalPostDX = gs−>GoalOppPostDoX();
    double goalPostDY = gs−>GoalOppPostDoY();

    dist = brankaX − rs.Position().x;
    dx = predictedBallX − rs.Position().x;
    dy = predictedBallY − rs.Position().y;
    de = sqrt(dx∗dx + dy∗dy);

    if (XY_to_Cm(dist) <= veryClose && predictedBallX > rs.Position().x && XY_to_Cm(de) <
        maximumHitDistance) // zda je robot blizko brany soupere && zda mic je pred robotem &&
        zda mic je blizko robota
    {
        if (rs.Position().y > goalPostUY && rs.Position().y < goalPostDY) return true; // robot je
            mezi tycemi
        else if (rs.Position().y < goalPostUY)
        {
            dx = goalPostUX − rs.Position().x;
            dy = goalPostUY − rs.Position().y;
            angle = (180/M_PI) ∗ atan2(dy,dx);
            if (angle <= maxShotSpinAngle && angle >= 0 && predictedBallY >= rs.Position().y)
                return true; // robot je mimo horni tyc ale jeste pod streleckym uhlem && mic musi byt
                pred branou(ne smerem do rohu)
            else return false;
        }
        else if (rs.Position().y > goalPostDY)
        {
            dx = goalPostDX − rs.Position().x;
            dy = goalPostDY − rs.Position().y;
            angle = (180/M_PI) ∗ atan2(dy,dx);
            if (angle >= −maxShotSpinAngle && angle <= 0 && predictedBallY <= rs.Position().y)
                return true;// robot je mimo dolni tyc ale jeste pod streleckym uhlem && mic musi byt
                pred branou(ne smerem do rohu)
            else return false;
```

```
        }
    }
    else return false;
}
```

Výpis 20: Výpočet pro vlastnost Střela z otočky

```
bool RobotFeatures::PredictedShot(RobotSoccer& rs, Storage& st)
{
    // temp veci
    int robotX = rs.Position().x;
    int robotY = rs.Position().y;
    int robotRotation = rs.Rotation();
    int ballPX = st.Ball().PredictedPosition().x; //predicted ball
    int ballPY = st.Ball().PredictedPosition().y;
    int goalPostUX = gs->GoalOppPostUpX();
    int goalPostDX = gs->GoalOppPostDoX();
    int goalPostUY = gs->GoalOppPostUpY();
    int goalPostDY = gs->GoalOppPostDoY();
    //zustanou
    double dx,dy;
    double alfa,tempAngle1,tempAngle2,temp;

    dx = ballPX - robotX;
    dy = ballPY - robotY;

    alfa = (int)(180/M_PI * atan2(dy,dx)); //uhel robota k mici
    temp = alfa - robotRotation;
    while(temp > 180) temp -= 360;
    while(temp < -180) temp += 360;

    if (temp < maxPredictedShotAngle && temp > -maxPredictedShotAngle) //natoceni k mici (
        konstanta 25)
    {
        dx = goalPostUX - robotX;
        dy = goalPostUY - robotY;
        tempAngle1 = (int)(180/M_PI * atan2(dy,dx)); // vypocet uhlu k horni tyci

        dx = goalPostDX - robotX;
        dy = goalPostDY - robotY;
        tempAngle2 = (int)(180/M_PI * atan2(dy,dx)); //vypocet k dolni tyci

        if (alfa >= tempAngle1 && alfa <= tempAngle2) return true;
        else return false;
    }
    else return false;
}
```

Výpis 21: Výpočet pro vlastnost Trojúhelník

```
bool RobotFeatures::KickAway(RobotSoccer& rs, Storage& st)
{
    // temp veci
    int robotX = rs.Position().x;
```

```
    int robotY =rs.Position().y;
    int robotRotation = rs.Rotation();
    int ballX = st.Ball().Position().x;
    int ballY = st.Ball().Position().y;
    //zustanou
    double dx;
    double dy;
    double de,theta_e,angle;

    dx = ballX - robotX;
    dy = ballY - robotY;
    de = sqrt(dx*dx + dy*dy);

    angle = (180/M_PI) * atan2(dy,dx);
    theta_e = angle - robotRotation;
    while (theta_e > 180) theta_e -= 360;
    while (theta_e < -180) theta_e += 360;
    de = XY_to_Cm(de);

    if (angle > -90 && angle < 90)
    {
        GlobalFeatures* GF = new GlobalFeatures();
        if (GF->GetMicNaNasiStrane() && de < maxInRange && theta_e < maxAngle && theta_e >
            -maxAngle) return true; // jeste by mnel byt zavisly na rychlosti mice
        else return false;
    }
    else return false;
}
```

Výpis 22: Výpočet pro vlastnost Odkop

```
bool RobotFeatures::GoalMove(RobotSoccer& rs, Storage& st)
{
    int goalCentrX = gs->GoalHomeCentreX();
    if (abs(goalCentrX - st.Ball().Position().x) < XY_to_Px(farDistance)) // prevod na Px
    {
        return true;
    }
    else
        return false;
}
```

Výpis 23: Výpočet pro vlastnost Pohyb brankáře

```
void Skills::Position(RobotSoccer& rs, double x, double y, bool hasBall)
{
    int desired_angle=0,theta_e=0;
    double dx,dy,de,Kp=0.8,Ka=0.03;

    dx = x - rs.Position().x;
    dy = y - rs.Position().y;

    de = sqrt(dx * dx + dy * dy);
    if (dx == 0 && dy == 0)
```

```
{
    desired_angle = 90;
}
else
{
    desired_angle = (int)(180/M_PI * atan2(dy,dx));
}
theta_e = desired_angle - rs.Rotation();
while(theta_e > 180)
{
    theta_e -= 360;
}
while(theta_e < -180)
{
    theta_e += 360;
}

de = rf->XY_to_Cm(de); // vypocet realne vzdalenosti de

if (theta_e < 75 && theta_e > -75)
{
    if (de > 100)
        Ka = (5/2)/75;
    else if (de > 80)
        Ka = (9/2)/75;
    else if (de > 60)
        Ka = (13/2)/75;
    else if (de > 40)
        Ka= (17/2)/75;
    else
        Ka = (20/2)/75;

    if (de < 25)
    {
        Kp = 1;
        de = 25;
    }
    rs.VelocityLeft ((int)(Kp * de + Ka * theta_e));
    rs.VelocityRight ((int)(Kp * de - Ka * theta_e));
}
else
{
    if (hasBall) // nemuze se otoci na miste jelikoz by prisel o mic
    {
        if (theta_e > 0)
        {
            rs.VelocityLeft(15);
            rs.VelocityRight(5);
        }
        else
        {
            rs.VelocityLeft(5);
            rs.VelocityRight(15);
        }
```

```
        }
        else
        {
            rs.VelocityLeft((int)(Ka * +theta_e));
            rs.VelocityRight((int)(Ka * -theta_e));
        }
    }
}
```

Výpis 24: Výpočet rychlostí robota pro přímé přemístění

```
void Skills::ObstacleAvoidance( RobotSoccer& rs, Storage& st,
    int positionX, int positionY, bool hasBall)
{
    double dist, length, angle, theta_d, diff_angle,
        tmp_x, tmp_y, temp=0, theta_e, Ka=0.05, Kp = 0.8;
    int jj, ii;
    // distance between robot and obstacle
    for(int i=0;i<gs->NUMBER_OF_ROBOTS();i++)
    {
        for(int j=0;j<2;j++)
        {
            if(j==0)
            {
                tmp_x = st.MyRobots()[i].Position().x;
                tmp_y = st.MyRobots()[i].Position().y;
                if(rs.Position().x == tmp_x && rs.Position().y == tmp_y) continue;
                else if((tmp_x > rs.Position().x && tmp_x < positionX) ||
                    (tmp_x > positionX && tmp_x < rs.Position().x)) continue;
                else
                {
                    dist = sqrt((rs.Position().x-tmp_x) * (rs.Position().x-tmp_x) +
                        (rs.Position().y-tmp_y) * (rs.Position().y-tmp_y));
                }
            }
            else
            {
                tmp_x = st.OppRobots()[i].Position().x;
                tmp_y = st.OppRobots()[i].Position().y;
                if((tmp_x > rs.Position().x && tmp_x < positionX) ||
                    (tmp_x > positionX && tmp_x < rs.Position().x)) continue;
                else
                {
                    dist = sqrt((rs.Position().x-tmp_x) * (rs.Position().x-tmp_x) +
                        (rs.Position().y-tmp_y) * (rs.Position().y-tmp_y));
                }
            }
            if(temp == 0) temp = dist;
            else if(dist < temp) // vyhledam nejblizsiho robota - jj, ii urcuje o ktereho se jednalo
            {
                temp = dist;
                jj = j;
                ii = i;
```

```
        }
    }
}
if ( jj ==0)
{
    tmp_x = st.MyRobots()[ii].Position().x;
    tmp_y = st.MyRobots()[ii].Position().y;
}
else
{
    tmp_x = st.OppRobots()[ii].Position().x;
    tmp_y = st.OppRobots()[ii].Position().y;
}
dist = temp;
angle = atan2(tmp_y - rs.Position().y, tmp_x - rs.Position().x);
theta_d = atan2(positionY - rs.Position().y, positionX - rs.Position().x);
diff_angle = theta_d - angle;
length = temp * sin(abs(diff_angle));

while( diff_angle > M_PI ) diff_angle -= 2.* M_PI;
while( diff_angle < -M_PI ) diff_angle += 2.* M_PI;

if ( (length < obstacleArea+modificationArea) && (abs(diff_angle )< M_PI/2)) // kdz je moc
     blizko presto do smeru nemusi zasahovat
   // V podstate z cestou prichazi nejblize do styku az za robotem, je potreba vytahnou uhel
        90 stupnu
{
    if ( dist <= obstacleArea ) theta_d = angle - M_PI; // robot jede na opacnou stranu nez je
        jeho protivnik
   else if ( dist <= obstacleArea + modificationArea )
   {
       // upravi theta_d po smeru hodin
       if ( diff_angle > 0 || (gs->GameFieldDoY() < (tmp_y + obstacleArea + modificationArea
           ))) // na kladne Y ose scitani +
       {
           tmp_x = ( (dist-obstacleArea)*cos(angle+1.5*M_PI) +
               (obstacleArea+modificationArea-dist)*cos(angle+M_PI) ) / modificationArea;
           tmp_y = ( (dist-obstacleArea)*sin(angle+1.5*M_PI) +
               (obstacleArea+modificationArea-dist)*sin(angle+M_PI) ) / modificationArea;
           theta_d = atan2( tmp_y, tmp_x );
       }
       // upravi theta_d proti smeru hodin
       else if ( diff_angle <= 0 || (gs->GameFieldUpY() > (tmp_y - obstacleArea -
           modificationArea))) // na zaporne Y ose scinati -
       {
           tmp_x = ( (dist-obstacleArea)*cos(angle+0.5*M_PI) +
               (obstacleArea+modificationArea-dist)*cos(angle+M_PI) ) / modificationArea;
           tmp_y = ( (dist-obstacleArea)*sin(angle+0.5*M_PI) +
               (obstacleArea+modificationArea-dist)*sin(angle+M_PI) ) / modificationArea;
           theta_d = atan2( tmp_y, tmp_x );
       }
   }
   else
   {
```

```
        // uprava theta_d po smeru hodin
        if ( diff_angle > 0 || (gs->GameFieldDoY() < (tmp_y + obstacleArea + modificationArea
             )))
        {
            theta_d = abs( atan( (obstacleArea+modificationArea) / sqrt(
                dist*dist - (obstacleArea+modificationArea)*(obstacleArea+modificationArea) )) +
                    angle;
        }
        // uprava theta_d proti smeru hodin
        else if ( diff_angle <= 0 || (gs->GameFieldUpY() > (tmp_y - obstacleArea -
            modificationArea)))
        {
            theta_d = -abs( atan( (obstacleArea+modificationArea) / sqrt(
                dist*dist - (obstacleArea+modificationArea)*(obstacleArea+modificationArea) )) +
                    angle;
        }
    }
}
else
{   // Nejblizsi prekazka mu nestoji v ceste - robot tedy pocita se vzdalenosti k cili
    tmp_x = positionX - rs.Position().x;
    tmp_y = positionY - rs.Position().y;
    dist = sqrt(tmp_x*tmp_x + tmp_y*tmp_y);
}
theta_d = 180/M_PI * theta_d; // rotace robota je ve stupnich
theta_e = theta_d - rs.Rotation();

while(theta_e > 180)
{
    theta_e -= 360;
}
while(theta_e < -180)
{
    theta_e += 360;
}

dist = rf->XY_to_Cm(dist); // vypocet realne vzdalenosti dist

if (theta_e < 75 && theta_e > -75)
{
    if ( dist > 100)
        Ka = (5/2)/75;
    else if ( dist > 80)
        Ka = (9/2)/75;
    else if ( dist > 60)
        Ka = (13/2)/75;
    else if ( dist > 40)
        Ka= (17/2)/75;
    else
        Ka = (20/2)/75;

    if ( dist < 15)
    {
        Kp = 1;
```

```
            dist = 15;
        }
        rs.VelocityLeft((int)(Kp * dist + Ka * theta_e));
        rs.VelocityRight((int)(Kp * dist - Ka * theta_e));
    }
    else
    {
        if(hasBall) // nemuze se otocit na miste jelikoz by prisel o mic
        {
            if(theta_e > 0)
            {
                rs.VelocityLeft(15);
                rs.VelocityRight(5);
            }
            else
            {
                rs.VelocityLeft(5);
                rs.VelocityRight(15);
            }

        }
        else
        {
            rs.VelocityLeft((int)(Ka * +theta_e));
            rs.VelocityRight((int)(Ka * -theta_e));
        }
    }
}
```

Výpis 25: Výpočet rychlostí robota pro nepřímé přemístění

```
void Skills::GoalKeeperPosition(RobotSoccer& rs, Storage& st, bool centre)
{
    int goalPUY = gs->GoalHomePostUpY();
    int goalPDY = gs->GoalHomePostDoY();
    int goalCenterY = gs->GoalHomeCentreX();
    int goalCenterX = gs->GoalHomeCentreY();
    double dx,dy,de,desired_angle=0,theta_e=0;
    int x,y;
    x = goalCenterX + rf->XY_to_Px(goalKeeperX); // robot se bude pohybovat pred brankou na X
        souradnici
    if(centre)
        y = goalCenterY;
    else
    {
        y = st.Ball().Position().y;
        if(y < goalPUY) y = goalPUY;
        else if(y > goalPDY) y =goalPDY;
    }

    dx = x - rs.Position().x;
    dy = y - rs.Position().y;

    de = sqrt(dx * dx + dy * dy);
```

```
    if (dx == 0 && dy == 0)
    {
        desired_angle = 90;
    }
    else
    {
        desired_angle = (int)(180/M_PI * atan2(dy,dx));
    }

    theta_e = desired_angle - rs.Rotation();
    while(theta_e > 180)
    {
        theta_e -= 360;
    }
    while(theta_e < -180)
    {
        theta_e += 360;
    }

    de = rf->XY_to_Cm(de);
    if (de < 0.5)
    {
        Stop(rs);
    }
    else if (theta_e < 5 && theta_e > -5)
    {
        rs.VelocityLeft(25);
        rs.VelocityRight(25);
    }
    else if ((theta_e > 175 && theta_e <= 180) || (theta_e >= -180 && theta_e < -175))
    {
        rs.VelocityLeft(-25);
        rs.VelocityRight(-25);
    }
    else
    {
        rs.VelocityLeft ((int)(0.5 * de + 0.04 * theta_e));
        rs.VelocityRight ((int)(0.5 * de - 0.04 * theta_e));
    }
}
```

Výpis 26: Výpočet rychlostí brankáře

```
void Skills::Shoot(RobotSoccer& rs, Storage& st, bool fast)
{
    double dx,dy,desired_angle=0,theta_e=0,Ka = 0.05;

    dx = st.Ball().Position().x - rs.Position().x;
    dy = st.Ball().Position().y - rs.Position().y;

    desired_angle = (180/M_PI) * atan2(dy,dx);

    theta_e = desired_angle - rs.Rotation();
    while(theta_e > 180)
```

```
    {
        theta_e -= 360;
    }
    while(theta_e < -180)
    {
        theta_e += 360;
    }

    if (fast)
    {
        rs.VelocityLeft(80 + Ka * theta_e);
        rs.VelocityRight(80 - Ka * theta_e);
    }
    else
    {
        rs.VelocityLeft(50 + Ka * theta_e);
        rs.VelocityRight(50 - Ka * theta_e);
    }
}
```

Výpis 27: Výpočet rychlostí pro střelu na bránu

```
void Skills::SpotTurn(RobotSoccer& rs,int x, int y)
{
    double dx,dy,angle,theta_e;
    dx = x - rs.Position().x;
    dy = y - rs.Position().y;
    angle = (180/M_PI) * atan2(dy,dx);
    theta_e = angle - rs.Rotation();
    while(theta_e > 180) theta_e -= 360;
    while(theta_e < -180) theta_e += 360;

    if (theta_e > 0)
    {
        rs.VelocityLeft(20);
        rs.VelocityRight(-20);
    }
    else
    {
        rs.VelocityLeft(-20);
        rs.VelocityRight(20);
    }
}
```

Výpis 28: Výpočet rychlostí pro otáčení na místě